DE MONTFORT
UNIVERSITY

An Intelligent Controller for Synchronous Generators

JEEN GHEE KHOR

NOVEMBER
1999

A thesis submitted in partial fulfilment of the requirements of De Montfort University for the degree of Doctor of Philosophy

ABSTRACT

This thesis presents the research work carried out in the area of design, simulation and implementation of a system for operating diesel-generators at variable speed. Recent developments in control systems for synchronous generators and Electronic Design Automation (EDA) techniques are reviewed and the requirements for a variable speed generator system established. The basic system described is an as-dc-ac converter configuration comprising a rectifier, d.c. link and a three phase IGBT inverter. To validate the proposed system, a plant model of the diesel engine, synchronous generator and rectifier is derived. The model is developed in the same EDA environment as the control system to achieve a highly integrated design process. A complete analysis of PWM inverter system, including simulations performed in TCAD (a power electronic simulation software), as well as the design and hardware construction of PWM inverter control circuitry are presented.

The variable speed system is controlled using a hardware-based Fuzzy Logic Controller (FLC). The fuzzy system is designed and verified using a hardware description language (VHDL) and the design is implemented in a Xilinx XC4010 Field Programmable Gate Array (FPGA). Several original techniques, including the *mini*-Fuzzy Associative Memory (FAM) table technique, are devised to optimise the design within the constraints of the FPGA's area space. Finally, the complete variable speed control system is tested on a 7.5kW permanent magnet synchronous generator set. Results are presented which show that the control system is successful in governing the d.c. link voltage when subject to varying load conditions. The use of VHDL and other EDA tools in the design, simulation and implementation process are shown to offer substantial benefits, for example, reduced concept-to-implementation timeframe, greater design flexibility and higher rate of right-first-time designs. The intelligent controller devised successfully allows variable speed operation of the generator while maintaining the desired output frequency.

ACKNOWLEDGEMENTS

I would like to extend my deep gratitude to my supervisor, Professor Malcolm McCormick, for his invaluable guidance and support throughout the course of my research. I am also greatly indebted to Dr. Marcian N. Cirstea for his excellent advice both as a second supervisor and as a friend.

The work presented in this thesis would not have been possible without the much appreciated support from Professor Lawrence Haydock and the staff of Newage International.

Special thanks are due to Mrs. Sheila Hayto, Electronic and Electrical Engineering Departmental Secretary as her help and advice have proven to be invaluable assets in many situations. I must also acknowledge the stimulating technical discussions I have enjoyed with Dr. Wei Foong Low and the exceptional team work from all my colleagues in the Electrical Machines, Drives, Control Systems and Numerical Field Computation Research Group, especially Andrei Dinu and Yan-Ting Hu. To Dr. Richard Wilson, Dilip Chauhan, Pritesh Karia, Tim O'Mara and Steve Regester many thanks are due for their help in various aspects of the project.

I also wish to thank the people at Durham University who made it possible for me to conduct the practical experiments, namely Professor Ed Spooner, Dr. Jim Bumby and Mark Glendinning.

Last, but not in anyway least, I am particularly grateful to my parents, Ir. and Mrs. Khor Bok Chim for their unfailing support and encouragement. It is to them that I owe the most.

CONTENTS

Chapter
1

Introduction

Synchronous generators are responsible for the bulk of the electrical power generated in the world today. They are mainly used in power stations and are predominantly driven either by steam or hydraulic turbines. These generators are usually connected to an *infinite bus* where the terminal voltages are held at a constant value by the 'momentum' of all the other generators also connected to it. Another common application of synchronous generators is their use in stand alone or isolated power generation systems. The prime mover in such applications is usually a diesel engine.

The aim of the research presented in this thesis is to develop an improved control system for diesel engine driven stand alone synchronous generator sets. Its primary objective is to design and build a working prototype that incorporates a new control strategy and the latest engineering innovations. The subsidiary objectives include ensuring that the prototype system:

- can be used effectively as a starting point for further studies into a new generation of controllers for stand alone synchronous generator sets
- incorporates a certain amount of artificial intelligence such that it is flexible and not specific to a particular type of engine-generator set
- is designed using a systematic process which enables rapid prototyping of future improvements
- takes advantage of modern digital electronic technology.

1.1 Overview of the Project

A block diagram of the proposed system is shown in *Figure 1-1*. It consists of a permanent magnet synchronous generator, a rectifier, an inverter and two electronic controllers. A diesel engine is used to provide the mechanical power to the generator and acts as the prime mover.

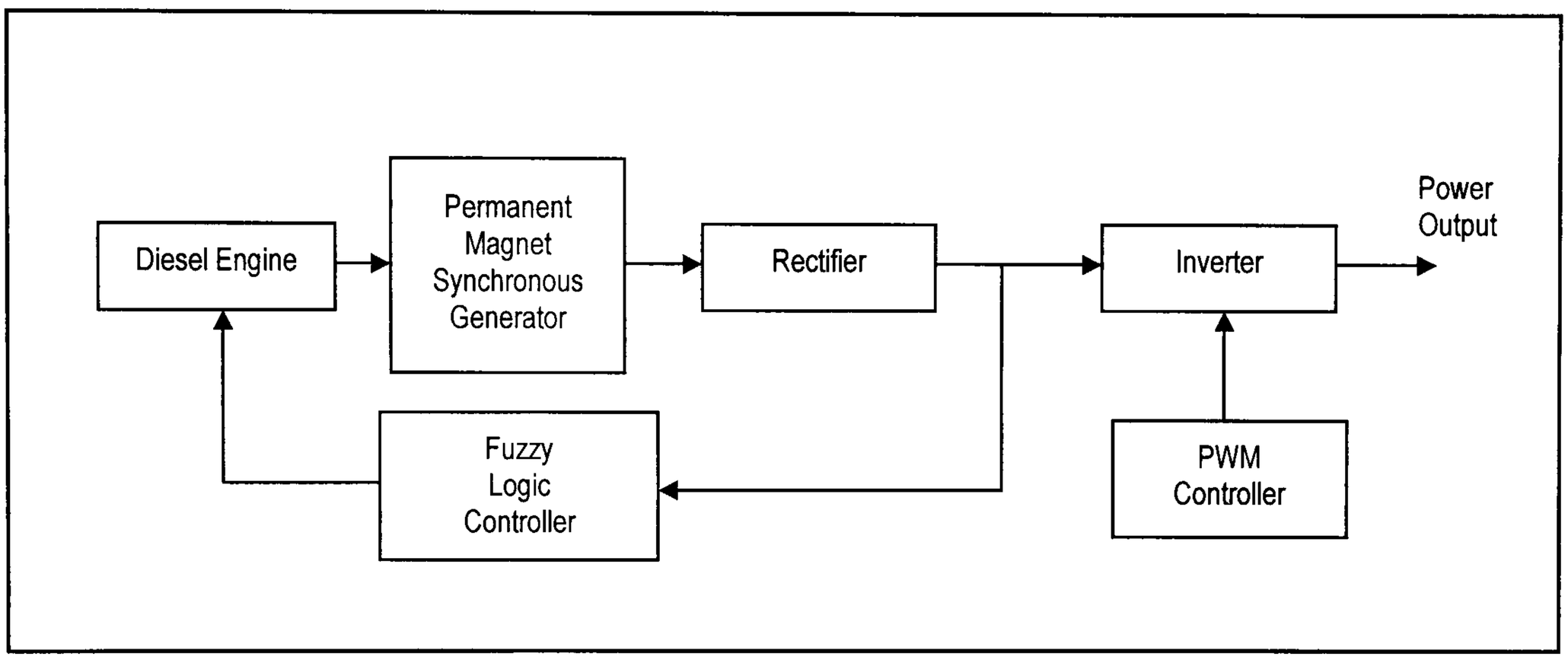


Figure 1-1 *Block diagram of the proposed generator control system.*

In a conventional electromagnet generator system, the generator output voltage is regulated via its field winding by an *automatic voltage regulator* while the speed of the engine is maintained at a fixed operational speed by a *speed governor*. Assuming a direct coupling or 1:1 gear ratio, the engine's operational speed is determined by the equation:

$$n_{mech} = \frac{f}{p}$$

where

n_{mech} is the mechanical operational speed in revolutions per second

f is the desired electrical frequency (usually 50Hz in U.K. and Europe)

p is the number of pole-pairs in the synchronous machine.

It may seem odd that the operational speed of the system should be determined by these variables instead of factors which have a more direct influence on the system's performance such as the engine's efficiency. Historically, and even in a lot of present

situations, this approach is viewed to be more economically and technically sound, since changing power frequencies can be difficult and expensive. However, in the past two decades or so, there has been a progressive development in the field of power electronic technology. With the introduction of power devices such as power Metal Oxide Semiconductor Field Effect Transistors (MOSFETs), Insulated Gate Bipolar Transistors (IGBTs), inverter and rectifier modules, the control of electrical power, especially at low voltages (415V a.c. and less) is becoming increasingly easy. With this in mind, the thesis proposes an engine-generator control system that is designed in a fashion that allows variable speed operation. In order to implement this strategy, an a.c.-d.c.-a.c. conversion system is necessary to isolate the output electrical frequency from the effects of variable speed operation. This conversion system is represented by the rectifier and inverter blocks in *Figure 1-1* and is referred to as the *d.c. link converter* in the thesis. Because the output frequency is maintained at a specified value, the complete system is referred to as a Variable-Speed Constant-Frequency (VSCF) System. A Pulse Width Modulation (PWM) inverter controller is used to maintain control over the output power waveform, therefore the power frequency can be totally independent of the mechanical speed. This allows the engine to be operated at a speed that is most suitable for the overall system efficiency.

A Fuzzy Logic Controller (FLC) is developed to control the speed of the engine. The control surface of the FLC is determined by a set of rule base. By changing the control inputs and the rule base the FLC can effectively be used to implement a wide range of different control laws, allowing the system to be studied under different criteria. In order to further enhance the efficiency of the design process, the FLC is developed on a fully computerised Electronic Design Automation (EDA) platform. Using this approach, the FLC is made up of individual 'components' designed using a hardware description language known as '*Very high speed integrated circuit Hardware Description Language*' (VHDL). The components are then integrated together and imprinted onto a Field Programmable Gate Array (FPGA) to form a control chip (integrated circuit). The architecture of the FLC can be restructured quite easily to suit different needs, simply by changing the 'components' in its system. This approach gives the FLC an almost generic feature.

1.2 Overview of the Thesis

In presenting the work of the research, the following two main arguments are advanced.

- Due to recent developments in power electronic technology, it has become feasible to consider exploiting the potential of operating stand alone power generators at variable speed, or at least, at speeds which are not determined exclusively by the desired frequency and number of poles in the machine.

- The use of VHDL and FPGA/ASIC technology in the design of control systems for electrical machines and power electronic systems increases the efficiency of the design process, even to the extent of enabling *rapid manufacturing* of integrated control circuits.

The content of the thesis is divided into nine chapters, including this introductory chapter. *Chapter 2* briefly traces the development of synchronous generators and their control systems since their invention while *Chapter 3* introduces the EDA software, design tools and programming languages used in the research. In *Chapters 4* and *5*, mathematical studies of the synchronous generator and various other elements of the control system are presented.

Chapters 6 and *7* enter into a detailed discussion on the theoretical as well as practical aspects of the fuzzy logic controller. Practical experiments, tests and results of the complete system are presented in *Chapter 8*. In the final chapter of the thesis, *Chapter 9*, there is a general discussion on the research project and some conclusions are drawn. Also included in that chapter is an evaluation of the limitations of the work and further considerations to address these issues.

1.3 Original Contributions of the Thesis

The original work and achievements of the project presented in this thesis can be summarised by the following points:

i. The design and implementation of a power electronic system including the simulation and analysis of the PWM inverter using a simulation tool called TCAD. Two control circuits were built to generate the PWM switching patterns: one incorporating the use of C programs and memory circuits while the other uses an integrated circuit.

ii. The development of mathematical models for the engine, generator and rectifier. Simulations of the models were achieved using VHDL. This allows the functionality of the FLC design to be verified before progressing to the implementation stage.

iii. The design and simulation of an original FLC using VHDL for the purpose of controlling the engine-generator set. The FLC was implemented in a Xilinx FPGA.

iv. The development of original ideas and algorithms such as the model of Mamdami's fuzzy inference engine using switches and the 'window' (using '*Mini*-Fuzzy Associative Memory tables) method of fuzzy inference to reduce the area size of the FLC design.

v. The design of numerous digital and analogue circuits as interfaces for control purposes.

Chapter 2

Control of Synchronous Generators

This chapter begins by looking at the development of power generators from the time of their invention to the present. The following sections discuss the control systems of synchronous generators, including topics such as power electronics and modern control theory in an attempt to tie these developments to the present research work.

2.1 Power Generators and Synchronous Machines

The discovery of electromagnetism by Michael Faraday in 1831 led to the rapid development of electromagnet machines for converting mechanical energy into electricity. Within a few months of Faraday's announcement, an Italian scientist, Signor Salvator dal Negro, invented an electric generator in which a permanent magnet was pushed and pulled to provide the necessary motion. The first of the rotating electromagnet generators as we know today was invented by Hypolite Pixii in Paris. It was made public at a meeting of Academie des Science in 1832. Later that year, Pixii added a commutator to his machine to obtain direct current (d.c.) from the alternating current (a.c.) produced. Early electric generators, or *dynamos* as they are known, produced d.c. electric current on a small scale. They were used mainly for supplying electro-plating baths and later for providing power to arc lamps in lighthouses. The invention of light bulbs and steam-engine-driven generators in America by Thomas A.

Edison led to the commercial expansion of electric generation for lighting purposes towards the end of the 19th century. In the early days direct current was the preference, but when long distance transmission become necessary alternating current was found to be more suitable. Power transmission at high voltages is more economical and the voltage level of alternating current can be easily changed using transformers. By the second half of the 20th century, alternating current became almost universal, leading to the widespread use of a.c. generators. Among the various types of a.c. generators, the polyphase synchronous generator is the largest single-unit electrical machine in production today, with power ratings of up to several hundred MVA's being common. They are widely used in large power stations as well as in industrial, marine, telecommunication and other standby or continuous power applications. Recent work in synchronous generators is mainly aimed at improving the efficiency of the machine, quality of the output power and the stability of the system.

Although a massive proportion of synchronous generators are electromagnetic, the use of permanent magnet synchronous machines as stand alone generators has been studied for more than half a century. Permanent magnet synchronous generators (PMSGs) are more difficult to regulate and it is only with the recent developments in power electronics that they are seriously being considered for various applications [1, 2, 3]. One of the main advantages of the control system proposed in this thesis is its ability to regulate stand alone PMSGs as well as electromagnet generators. This functionality is duly demonstrated by the experiments presented in *Chapter 8* of this thesis, in which a PMSG is used.

It has to be mentioned that synchronous machines are by no means the only type of electrical machine used for stand alone power generation. Studies have been conducted into the use of induction generators [4, 5], reluctance generators [6] and other types of machines that might prove to be more suitable in certain applications.

2.2 Power Electronics

Since the invention of electrical machines in the 19th century, there has been a need to convert electrical power for various applications such as electrical machine drives, voltage regulation, welding, heating, etc. Initially, rotating machines were predominantly used to control and convert electrical power. It was the introduction of the glass bulb mercury arc rectifier (1900) which led to the beginning of the power electronics era. Power Electronics is the branch of engineering concerned with the application of electronics in the control and conversion of electrical power.

Early power electronic devices such as thyratrons and ignitrons were crude and unreliable. The introduction of selenium rectifiers during the World War II was particularly welcomed due to their reliability. In 1948, the invention of the p-n junction transistor by Bardeen, Brattian and Shockley from Bell Laboratories was seen as a revolutionary advancement in the field of electronics. This laid the foundation for the development of the p-n-p-n transistor switch by J.L. Moll, *et. al.* (1956), a device which later became known as the thyristor, or silicon controlled rectifier (SCR). By 1957, the first commercial thyristor was made available by General Electric Company. This marked the beginning of modern power electronics era. This three terminal device had a continuous current rating of 25A and a blocking voltage of up to 300V. Since then, the thyristor has become one of the most popular devices in power electronics. Circuit design engineers have constantly worked on improving the operating performance of the thyristor, resulting in the creation of a range of different types of thyristors optimised for different applications. They can generally be grouped into six categories, namely [7]:

1. Phase Control Thyristor
2. Inverter Thyristor
3. Asymmetrical Thyristor
4. Reverse Conducting Thyristor (RCT)
5. Gate-Assisted Turn-Off Thyristor (GATT)
6. Light-Triggered Thyristor.

Another class of power electronic devices subsequently developed were the *controllable power switches*. Thyristors, while being able to be latched on by a control signal, can only be turned off by the power circuit, which is a great drawback. However, controllable switches can be turned on as well as turned off by the control signals. Although controllable switches like the transistor have been around since 1948, designing them to possess high power handling capabilities was not achieved until much later. Compared to thyristors, controllable power switches offer greater flexibility in power applications, including the possibility of controlling d.c. circuits without complicated commutation circuitry. Thus, they are particularly attractive in inverter applications. Examples of devices in this category are Gate Turn-off Thyristors (GTO), power transistors, power MOSFETs , Integrated Gate-Commutated Thyristors (IGCT) and IGBTs.

The GTO is a thyristor-like latching device but can be turned off by a negative gate current. Power transistors and power MOSFETS were developed from small signal designs to later versions which are capable of handling higher voltage applications in the order of hundreds of volts. In the early 1980s, the insulated gate bipolar transistor (IGBT), which combines the low on-state conduction losses of the bipolar junction transistor (BJT) and the high switching frequency of the MOSFET, was developed [8] . The IGBT has since gained widespread popularity in power electronic applications. Commercial IGBTs are currently available up to 3.3kV. These components can be utilised in a range of power applications. The development of such power devices is expected to grow as the use of new materials such as monocrystaline silicon carbide (SiC) increases their voltage ratings and reduces thermal resistance [9, 10].

Generally, a power electronic system comprises two separate sets of circuits:

- the logic level control circuitry
- the high power circuits.

Recent developments in electronics made it possible to combine these two components into a single integrated circuit, the power integrated circuit (PIC). A PIC is defined by Thomas [11] as an integrated circuit which combines the logic level control and/or protection circuitry with power handling capability of supplying 1A and withstanding at

least 100V. With the current trend towards integrated solutions, this technology is receiving a substantial amount of attention. Integrated power electronic devices are seen as the solution for smaller and lower cost power electronic systems in the future.

2.3 Control Theory

The function of a control mechanism is to maintain certain essential properties of a system at a constant value under perturbations. Historical control systems which are simple but effective have been employed in water regulation and control of liquid level in wine vessels for centuries. Some of these concepts are still used today, for example the float system in the water tank of the toilet flush. However, modern control systems used in today's industry, including the control of industrial generators, are much more complex and owe their beginnings to the development of the Control Theory. The earliest significant work in modern automatic control can be traced to James Watt's design of the *fly-ball governor* (1788) for the speed control of a steam engine. In 1868, Maxwell [12] presented the first mathematical analysis of feedback control. It was during this time that systematic studies into control systems and feedback dynamics began. One significant development was the well known Routh's Stability Criterion (1877) which won E.J. Routh the Adam's Prize for that year.

Early 20th century saw the beginning of what is now known as **Classical Control Theory**. Minorsky's work (1922) on the determination of stability from differential equation describing the system (characteristic equation) and Nyquist's development (1932) of a graphical procedure for determining stability (frequency response) have largely contributed to the study of control theory. In 1934, Hazen [13] introduced the term 'servomechanism' to describe position control systems in his attempt to develop a generalised theory of servomechanism. Two years later, the development of the *Proportional-Integral-Derivative (PID) controller* was described by Callender et al. (1936). The Control Theory, like many branches of engineering underwent significant developments during World War II. Based on Nyquist's work, H.W. Bode introduced a method of feedback amplifier design, now known as the *Bode plot* (1945). By 1948,

root locus method of design and stability analysis was developed by W.R. Evans [14]. With the introduction of digital computers in the 1960s, the use of frequency response and characteristic equations began to give way to Ordinary Differential Equation (ODE), which worked well with computers. This led to the birth of **Modern Control Theory**. While the term Classical Control Theory is used to describe design methods of Bode, Nyquist, Minorsky and similar workers, Modern Control Theory relies on ODE design methods, like the *State Space Approach*, which are more suitable for computer aided engineering. Both these branches of control theory rely on mathematical representation of the control plant from which to derive its performance. To address the issues of non-linearities and time-variant parameters in plant models, control strategies that continuously adapt to the variations of plant characteristics have been introduced. Generally known as Adaptive Control Systems, they include techniques such as self-tuning control, H-infinity control, model referencing adaptive control and sliding mode control. Studies also include the use of non-linear state observers to continuously estimate the parameters of the control plant [15]. They can be employed to tackle the issue of *non-observability*, that is the condition whereby not all of the required states are available for feedback. This may be the cheaper solution because it does not require as many sensors, such as in variable speed drives [16], or because it is physically difficult or even impossible to obtain the feedback states such as in a nuclear reactor.

In many instances, the mathematical model of the plant is simply unknown or ill-defined, leading to greater complexities in the design of the control system. It has been proposed that **Intelligent Control Systems** give a better performance in such cases. Unlike conventional control techniques, intelligent controllers are based on *Artificial Intelligence* (AI) rather than plant model. They imitate the human decision-making process and can often be implemented in complex systems with more success than conventional control techniques. AI can be classified into expert systems, fuzzy logic, artificial neural networks and genetic algorithms. With the exception of expert systems, these techniques are based on *soft-computing* methods. This means that they are capable of making approximations and 'intelligent guesses' where necessary, in order to come out with a 'good enough' result under a given set of constraints. Intelligent control systems may employ one or more AI techniques in its design.

2.4 Synchronous Generator Control

The study of synchronous generator control systems can roughly be divided into two parts: voltage regulation and speed governing. Both control elements contribute to the stability of the machine in the presence of perturbations. A reliable control system set is essential for the safe operation of generators. There are various methods of controlling a synchronous generator and suitability will depend on the type of machine, its application and the operating conditions. For instance, the voltage regulation of an electromagnet synchronous generator is usually achieved by controlling the field excitation current whereas permanent magnet generators do not have excitation systems and require a totally different strategy.

The voltage regulation system in a electromagnet synchronous generator is called an Automatic Voltage Regulator (AVR). It is a device which automatically adjusts the output voltage of the generator in order to maintain it at a relatively constant value. This is achieved by comparing the output voltage with a reference voltage and, from the difference (or *error*), it makes the necessary adjustments in the field current to bring the output voltage closer to the required value. Older AVRs used in the early days belong to a class of electromechanical devices. They are generally slow acting and possess zones of insensitivity known as *dead bands.* There is a wide variety of electromechanical AVRs, ranging from vibrating contact regulators to carbon pile regulators [17, 18]. However, they are now replaced with continuously-acting electronic regulators. These electronic AVRs are much faster and do not possess dead bands, hence the term 'continuously-acting'. *Figure 2-1* shows a block diagram of an example of an electronic AVR system [19]. The generator set comprises two sections: the Main section and an Exciter. Each section consists of an armature winding and a field winding. Electrical power is derived from the terminals of the Main armature winding. The AVR maintains a closed-loop control of the terminal voltage by taking in a ratio of the Main armature voltage as the input, comparing it to a pre-set reference value and producing a control signal to the Exciter field based on the *error*. This induces the correct amount of current in the Exciter armature and the current is transferred into the Main field winding via a set of rotating diodes. The complete system is an efficient and well established method of regulating the terminal voltage in a stand alone electromagnet synchronous

generator set. Power system generators also employ AVRs in their control systems, but they form only a part of a comprehensive control and regulation system.

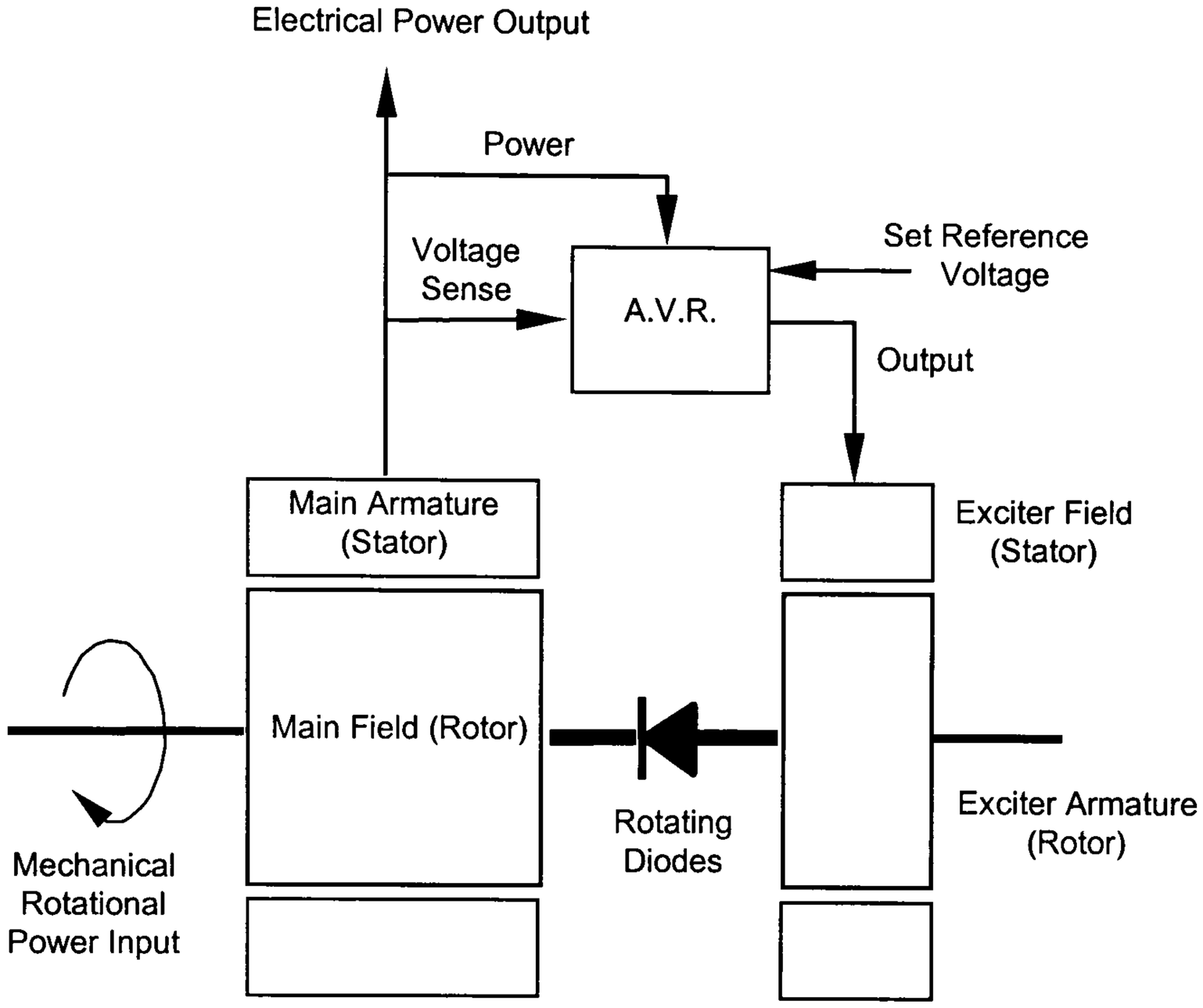


Figure 2-1 *Block Diagram of a Synchronous Generator and AVR.*

Most AVRs utilise analogue electronics to accomplish the control tasks. It is found to be cheap, simple and effective for the job. However, like all electronic applications, there is a growing interest in applying digital electronic technology to AVRs. The major advantages of digital AVRs over their analogue counterparts are the capability of realising sophisticated control functions and the ease of transmission and recording of information. The nature of digital systems allows complex algorithms to be executed either as a software based system [20] or a hardware based system [21]. Digital AVRs are capable of performing all the tasks of their analogue counterparts as well as a range of additional functions, including the following:

i. They can have direct interaction with higher levels of controls. This allows them to be interfaced with the main computer of the building, power plant or supervisory system, enabling *inter-operability*.

ii. The control parameters are more accessible to modifications with built-in logic switches and software tuning programmes. Changes in the control parameters of analogue AVRs can be difficult and expensive to implement. Therefore the parameter values of analogue AVRs are usually fixed at the design and commission stage.

iii. Stabilisers can be configured to switch in and out the system without interrupting the service, depending on the need.

iv. They can operate at optimal or sub-optimal conditions over a wider range of operation. This is especially true with the introduction of adaptive and intelligent controllers.

In as early as 1976, Malik, Hope and Huber worked on a software-based digital AVR [20] for a power system generator, whereby the control functions of the AVR were stored in an on-line station mini computer. The flexibility of a software based system extended the mathematical capabilities of the controller and allowed for one basic hardware design to be used with different control strategies, thus drastically reducing the cost of redesigning the controller to different specifications.

The digital AVR used by Hirayama, *et. al.* (1993) [22] was capable of all the functions of an analogue AVR as well as a range of additional features such as advanced fault status indication, self diagnostic features, verification of generator dynamics and recording of transients. A more comprehensive monitor of the power systems was also possible by feeding the recorded transient data into a personal computer (PC) via a RS232C link. Dedicated software in the PC will then calculate the Bode diagram and indicate transient responses and rise times. Similarly, the Digital Excitation Control Systems (DECS) used in Godhwani and Basler's (1996) [23] study into design methodology of controllers offer all the advantages of digital controls. In their work, the PID settings for a generator controller were programmed into an Electronically Erasable Programmable Read Only Memory (EEPROM). Due to the fact that different systems require different PID settings for optimal performance, manufacturers of analogue controllers often have to provide multiple designs to accommodate variations in stability networks. Godwhani and Basler proposed a design methodology - Direct Design Method of Controller Design - that allows the closed loop poles to be placed at any desired location. The PID settings were then custom designed for each individual

system using this method and stored into the EEPROM. This creates a single design which is flexible for the operation of a wide range of generator sizes (the study tested on generators ranging from 10kW to 50MW).

As a result of the highly non-linear nature of synchronous machines, it can be quite difficult to design a control system with high performance over different operating points. This is becoming an increasingly important issue especially in power systems. To address the problem, Marino [24] proposed *exact feedback linearisation*, a technique which received some recent attention [25]. There is also interest in adaptive AVRs [26, 27, 28] and power system controllers. There are currently several different approaches to adaptive control of synchronous generators in power system control such as:

- linear optimal control theory [29]
- self-tuning control theory [30, 31]
- fuzzy logic [32]
- adaptive neural-network [33, 34, 35]

Although these works are mainly targeted for power systems applications, some of the ideas presented can be incorporated into other applications. Stand alone synchronous generators, systems with conventional AVRs and speed governors have always been operated at a fixed speed. The speed is determined by the desired power frequency and the number of pole-pairs in the machine. Running the generator at any other speed is not usually considered as a design option. One of the main problems in such a scheme is the task of maintaining the desired power frequency, especially under heavy load. Since the frequency is directly proportional to the speed, any change in speed would certainly disrupt the shape of the power waveform in that the frequency would vary in relation to the speed. It is only with recent developments in power devices and converter technology that Variable-Speed Constant-Frequency (VSCF) systems have been given serious attention. The matrix converter proposed in [36] initiated several studies into the a.c.-a.c. converter [37, 38, 39], but disparate issues such as commutation problems and the complexity of switching schemes generate reservations for its use in the present work. A more common solution for a.c.-a.c. power frequency conversion is to buffer the transition with a d.c. link, effectively producing an a.c.-d.c.-a.c. conversion scheme. In

this thesis, a power electronic system implementing this scheme is called a *d.c. link converter*. The main components of a d.c. link converter include some form of a rectifier, energy storage component(s) such as a smoothing capacitor and an inverter. Some schemes also incorporate a boost converter in the d.c. link [40].

In the last decade, d.c. link converters received considerable attention in the area of VSCF wind energy systems. VSCF schemes are widely used in stand alone wind energy conversion systems to solve the problem of frequency fluctuations which result from changes in wind velocity and load. *Figure 2-2* shows a simplified block diagram of a typical VSCF wind energy conversion system that incorporates a d.c. link converter. Although most of these schemes are designed for induction generators [41,42,43,44], there are also numerous projects which involve synchronous generators [45]. Research in this area usually focus around control systems for capturing maximum energy from varying wind velocity. Others include work such as the study of interface systems to improve the quality of power from VSCF wind turbines connected to a utility grid [46,47,48].

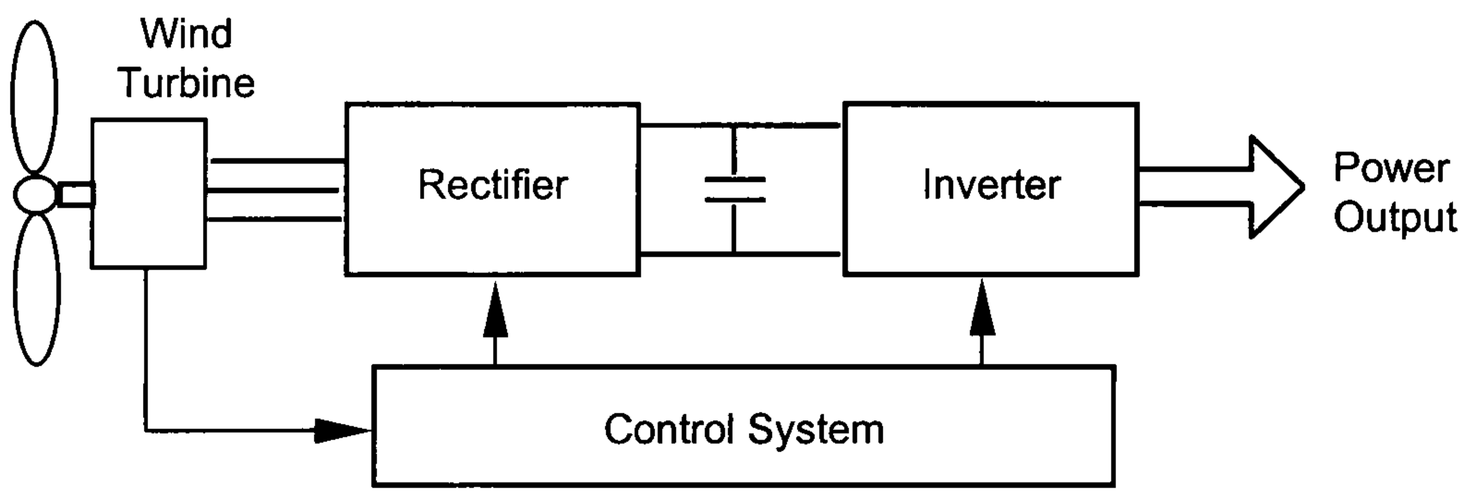


Figure 2-2 *Simplified block diagram of a VSCF wind energy conversion system.*

Unfortunately, the application of VSCF schemes in other systems such as stand alone generators is somewhat limited. In an engine driven system for example, the choice of speed can have great influence on the operational characteristics and efficiency of the engine. With the correct setting, it is possible for the system to be optimised for specific performances such as fuel consumption, exhaust emissions, vibrations and generator ratings. For example, in a domestic or recreational application, where the generator system has to be placed close to the living space, reducing noise pollution may be an important consideration when selecting the operational speed.

This chapter has shown how control theory and generator systems have evolved over the years. It is important to note that in building a control system, the design process can sometimes be just as important as the final product itself. The next chapter describes some of the electronic design tools used in the development process of the control system.

Chapter 3

Electronic Design Automation

With the increasingly competitive nature of the electronics industry, the development time for new products is rapidly decreasing. Engineers are constantly expected to develop new products for the market within a short time. The introduction of Electronic Design Automation in the late 1970s and early 1980s has allowed the development time of electronic designs to be shortened considerably. EDA is a design methodology in which dedicated tools, primarily software products, are used to assist in the development of integrated circuits, Printed Circuit Boards (PCBs) and electronic systems. In the early days, EDA tools were nothing more than a set of incoherent design tools that aid in a specific stage of the development, providing what are called *'islands of automation'*. Where the different tools need to share data, user-written data translators were sometimes used. EDA tools have since evolved into an integration of design toolsets that conform to standard data management protocol, thus eliminating the need for data translators.

Some of the advantages of EDA include[49] :

- enabling more thorough verification of design using simulation tools. This allows the design to be verified before being implemented into hardware, thus design faults can be detected in the early stages of the design process.
- exploring alternative designs using the synthesis and implementation tools. The designer can create a few alternative designs before selecting the best design for the implementation.

- automating some of the design steps, thus allowing the designer to concentrate on more important activities.
- ease in design data management.
- enabling the designer to operate at higher levels of abstraction, i.e. 'top-down' design method. This is possible with hardware description languages such as VHDL (see *Section 3.1*) and Verilog HDL. The designs are first described at register transfer level (RTL) where the design functions are addressed, with no reference to the hardware required for implementation. RTL descriptions can then be automatically translated into gate level using logic synthesis tools. This design methodology is similar to software programming, where the programme is written in a high level language before being converted into machine language.

The popularity of EDA tools increased rapidly with the widespread use of Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) in the 1980s. In ASIC technology, the cost of correcting a design flaw late in the design process can be very high. The need for 'right-first-time' designs led to demands for reliable EDA tools. With increasing use of ASICs and FPGAs in power electronic control systems, EDA techniques are increasingly being employed [50, 51, 52]. This has led to the development of a new design approach that relies more on verification by simulation, allowing new products to be developed and produced for the market in a shorter time.

3.1 VHDL

Hardware Description Languages (HDLs) in general were conceived as a result of the growing need to model and specify electronic circuits in a semantic form. HDLs share many common features with software programming languages but there are also distinct differences. For example, HDLs supports concurrent language construction while most software programs are based entirely on sequential operations. In addition, HDLs need to contain some structural information, which is not present in software programs. Detailed timing specifications are important in HDL description of circuits whereas software programs are usually less concerned with time-frames, with some exceptions. These differences result from the fundamental features of hardware circuits and software programs.

There are many types of HDLs in the market today and VHDL is arguably the most established one. VHDL is the acronym for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. It originated from the programming language ADA in the early 1980s as an offshoot of the VHSIC project by the United States Department of Defence. The aim of introducing this new language was to overcome two major drawbacks in the development of highly complex integrated circuit at the time: the lack of a language that could effectively describe the complex circuits which were being developed and the lack of a standard format among participating parties in the VHSIC programme. VHDL was adopted as an IEEE standard (VHDL-87) in 1987 and its development has been extensive ever since. Today, apart from circuit modelling, VHDL is also used in the design of electronic systems and forms an integral part of most EDA tools. In this project, VHDL is used to design a *fuzzy logic controller*. This section presents a brief description of the language for the benefit of the reader who is not familiar with VHDL. The materials presented here relates to the important features used in this thesis only. It is not a proper description of the language. The definition of the language is detailed in [53] and in-depth discussions can be found in all major VHDL text books [54, 55].

VHDL is a component-based language. Each VHDL *component* is made up of two main parts, an **entity** and an **architecture** as shown in *Figure 3-1*. The entity can be described as the outer shell of the component. It defines the input and output ports of the component and also describes how a component relates to the environment and other components. The actual functionality of the component is described in the architecture. It may contain a *behavioural-level* description, in which the design is modelled using mathematical functions or a *structural-level* description, in which the design is modelled using logical operations and possibly other components.

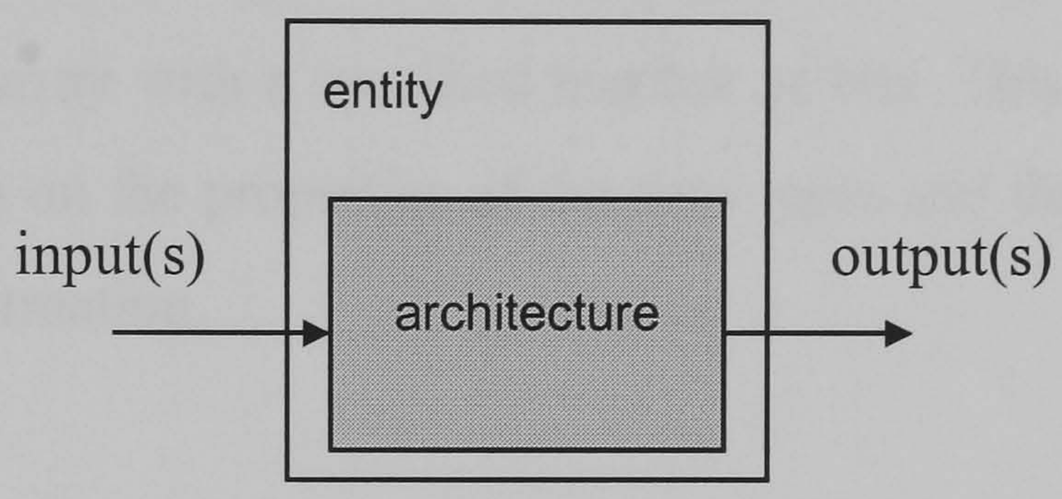


Figure 3-1 *VHDL Component.*

All statements within the VHDL architecture are executed concurrently. However, it is possible to define sequential operations but these operations can only exist within a **process**. It follows that, while the statements inside a process are executed sequentially, all processes inside an architecture are executed concurrently. Because VHDL supports both concurrent as well as sequential descriptions, it has become necessary to define two types of assignment operators. The symbol '**<=**' is the assignment operator for a concurrent object while '**:=**' assigns values to sequential objects. These operators are used throughout the designs presented in this thesis. Both can exist inside a process but like all sequential statements, the sequential assignment operator is illegal outside a process. It is worth noting that when concurrent statements are used inside a process, the effects of the statements are realised concurrently regardless of their positions within the process structure.

VHDL is a strongly typed language. The main data types used in this thesis are **real**, **integer** and **std_logic_vector**. The type **real** was used to declare objects for the mathematical simulations of the generator system described in *Chapter 4*. In the

Standard Library, it is specified as having the range from -1.0E+38 to +1.0E+38 [53], making it versatile for simulations. However, real objects do not have well-defined structural meaning and are not supported by current synthesis and implementation tools. Therefore, they are not used in the design of the systems which are implemented in hardware. The types **integer**, **std_logic** and **std_logic_vector**, on the other hand, are well supported for implementation.

Std_logic and **std_logic_vector** are data types that are declared in the IEEE package **std_logic_1164**. They are used extensively in the designs presented in *Chapter 7* because they represent actual hardware signals. Each **std_logic** object represents one bit and the **std_logic_vector** is an array with a specified number of bits. This thesis does not enter into detailed discussion on the properties of the data types and the reader is referred to [54, 55] for further information.

3.2 Xilinx Foundation Series

The Foundation Series is an EDA software by Xilinx Inc. for designing and implementing programmable hardware such as Field Programmable Gate Arrays (FPGAs) and Programmable Logic Devices (PLDs). It is made up of several EDA tools for specific design tasks which are discussed in greater detail later in the thesis. The main component of the software is the Foundation Project Manager, an application that manages all the different EDA tools in the software and maintains a unified environment for the user. *Figure 3-2* shows the window of the Project Manager. It comprises three frames. The upper right hand frame depicts the flow diagram of the project where the EDA tools are divided into five groups: Design Entry, Simulation, Implementation, Verification and Programming.

There are three EDA tools in the Design Entry group: HDL Editor, FSM (Finite State Machine) Editor and Schematic Editor. This allows the project design to be described either as a HDL program, a state machine description or as a schematic design. The design presented in this thesis is described using both the HDL Editor and the Schematic Editor. From the Design Entry stage, the design can be *synthesised*, a process that converts the design, whether it is a HDL program or a schematic, into a netlist

format. The netlists contains the structural description of the design and are used for various design operations including functional simulation. At this stage, it is not yet specific to any technology.

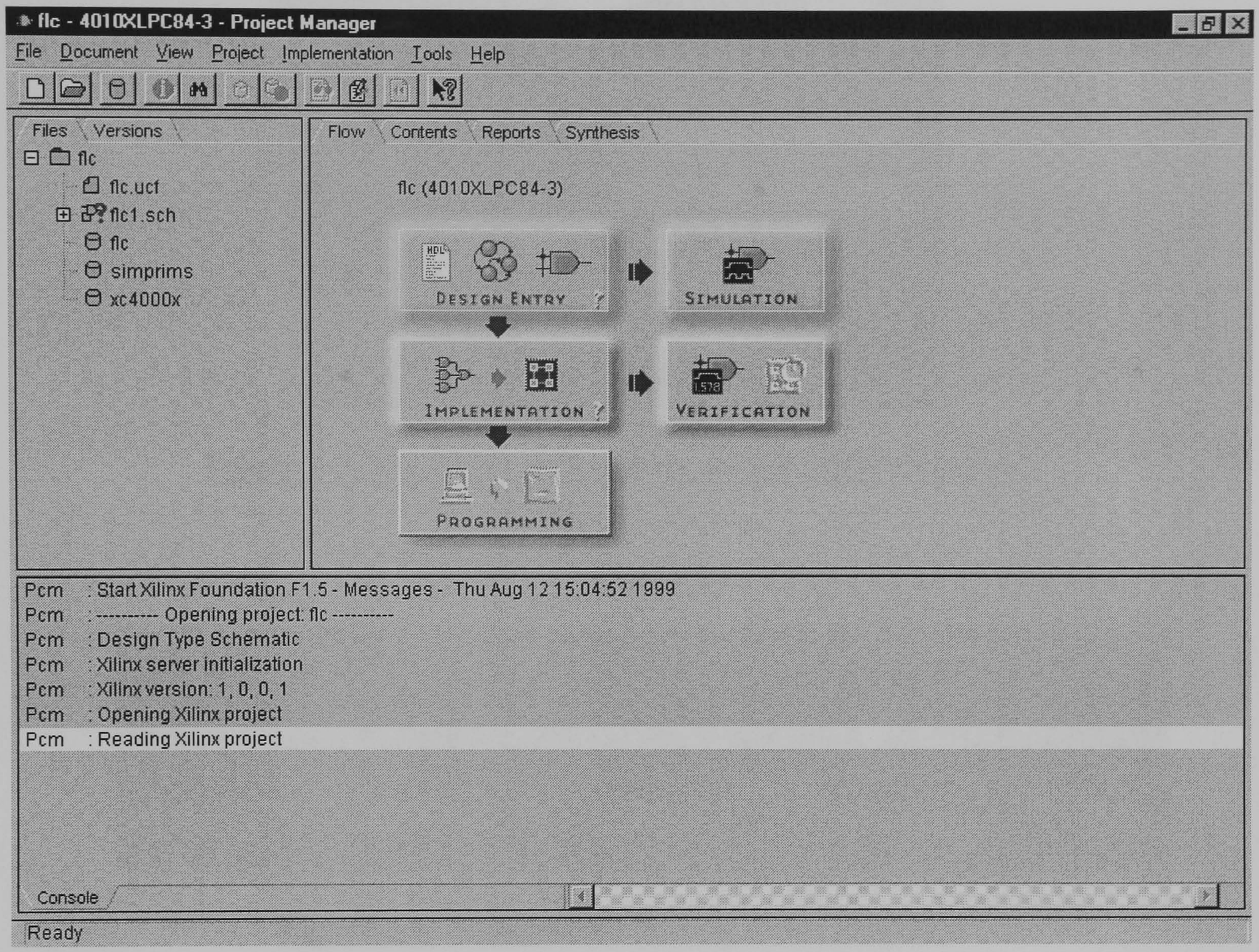


Figure 3-2 *Foundation Project Manager.*

In order to download the design into hardware, the target technology has to be specified. The netlist is complied into a format that is compatible to the targeted device in a process that is called *implementation*. It is important to note that the targeted device has to be confirmed at the start of the implementation procedure. *Figure 3-3* shows the implementation of a design targeted at the Xilinx XC4010XL-PC84 device. The segments of the implementation process are shown by the diagram in *Figure 3-4*. Further information on each implementation segments as well as on the Foundation Series in general can be found in [56, 57]. For the present discussion, it is sufficient to point out that the final product of this procedure is a bitstream file which can be directly downloaded into the targeted device.

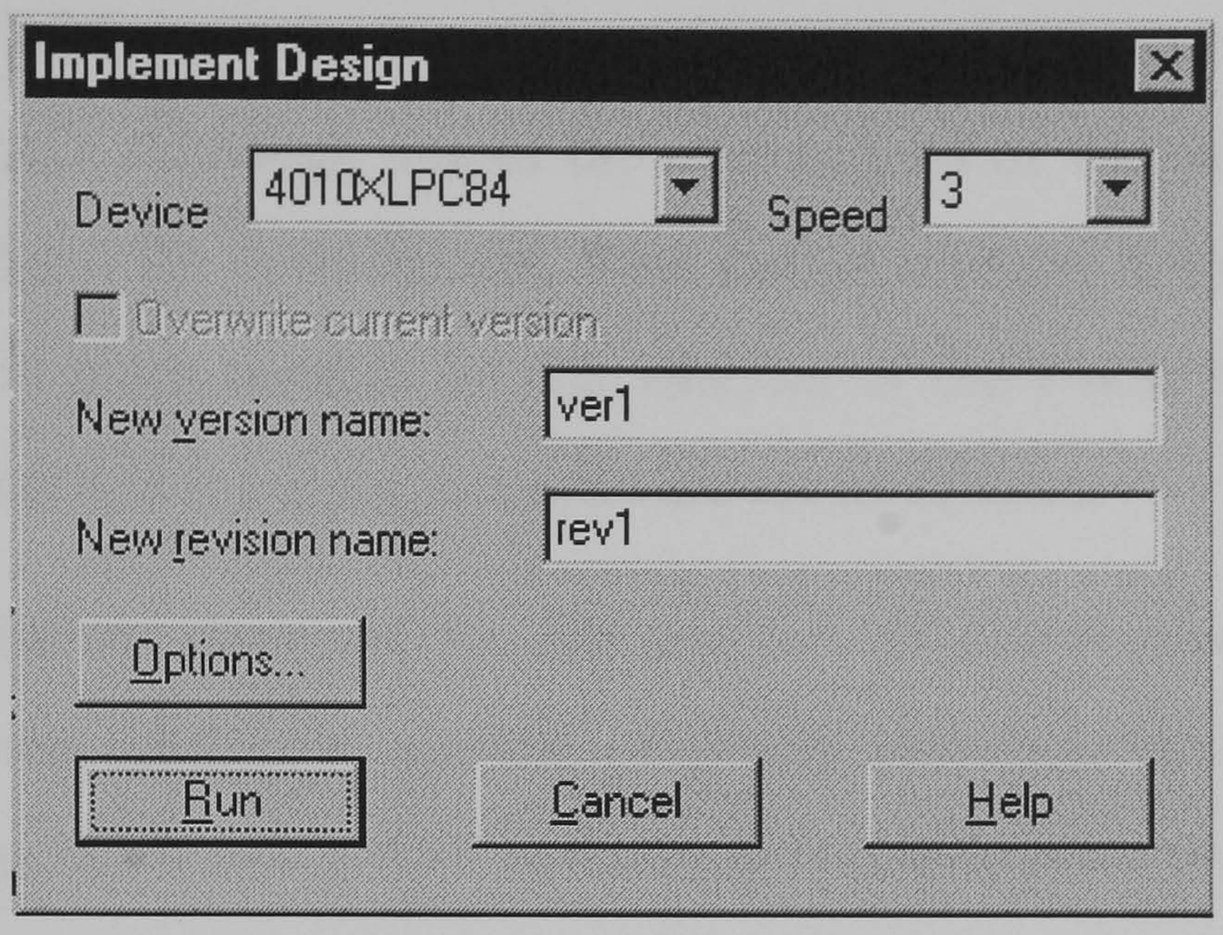


Figure 3-3 *Specifying target device during implementation.*

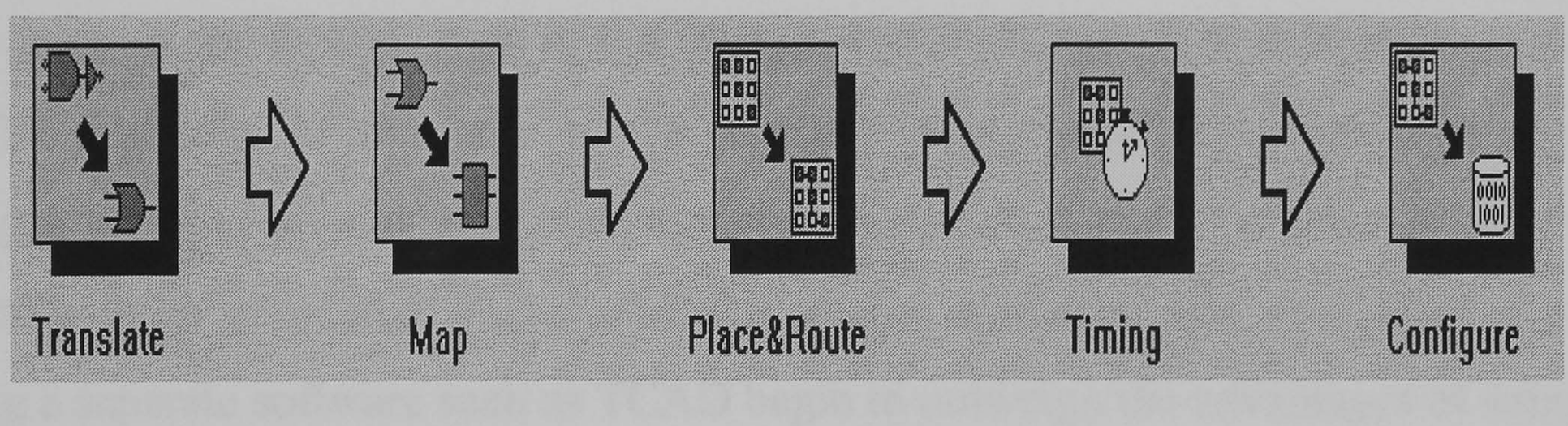


Figure 3-4 *Segments of implementation process.*

3.3 TCAD

Analysis of the PWM inverter is performed using a power electronic simulation software called TCAD. Although one of the propositions of this thesis is the use of a common EDA platform for development, from design through to implementation, it is found that TCAD is more suitable for the analysis of the PWM inverter for various reasons. The justifications for using this software are as follows:

- TCAD is specifically designed for the analysis of power electronic systems. It contains numerous predefined models of power electronic devices and electrical machines such as thyristors, power switches and synchronous generators. There are also logical function blocks to model control algorithms for the power electronic systems. Using a dedicated software such as TCAD eliminates the need to model certain objects from scratch and reduces the analysis time.

- The analysis and development of the PWM inverter are relatively independent of the FLC at this stage. Furthermore, unlike the FLC, the details of the inverter simulation designs are not required for circuit implementation. As a result of these factors, the benefits of an integrated design approach are not applicable and the advantages of using a separate software such as TCAD begin to outweigh the advantages of using a common EDA software, i.e. VHDL and Foundation Series.

Power electronic systems can usually be divided into two main sections, the power circuit and the control circuitry. The power circuit is made up of a set of devices with high power capabilities, for example IGBTs and thyristors, while the control circuitry is essentially the 'intelligent' part of the system and usually comprises low current devices and logic gates. TCAD is equipped to model and analyse both sections quite conveniently. The power circuit is analysed using circuit analysis which is based on Kirchhoff's laws. This method of computation is time consuming, therefore a different approach is used in the control section. The control circuitry can be adequately modelled by signal flow modelling which is based on dynamic input/output relations like transfer functions and Boolean algebra. Signals from the control circuitry can be directly applied to the power circuit. However, the variables of the power circuit have to be passed into the control circuitry via Sensor units. There are a list of predefined sensor units in the

library which can sense the voltage, current, torque, machine speed and even flux linkages from the power circuit.

Simulation of the power system can be performed either in MS DOS or in MS Windows. Waveforms are monitored during the simulation and recorded for further analysis using TCadGraph. A harmonic analysis of the recorded waveforms can also be obtained with TCadHar. It is possible to calculate harmonic analysis up to the 50^{th} order. These features make the software suitable for the analysis of the PWM inverter and control system presented in *Chapter 5*.

Chapter
4

System Representation

Before commencing an in-depth discussion about the control system, it is appropriate to first discuss the control plant, i.e. the elements to be controlled. In this chapter mathematical models of some of the control plant elements are described. As already mentioned in *Chapter 1*, the plant comprises a power electronic system, a permanent magnet synchronous generator and a diesel engine. Two control units are presented in this thesis, a Fuzzy Logic Controller (FLC) for the engine and a PWM controller for the output inverter. The analysis of the system is also divided into two parts as shown in *Figure 4-1*. Ensuing sections in this chapter discuss the control plant elements relating to the fuzzy logic controller, namely the rectifier, the generator and the diesel engine. The analysis of the inverter and the PWM controller is presented as a separate study in *Chapter 5*.

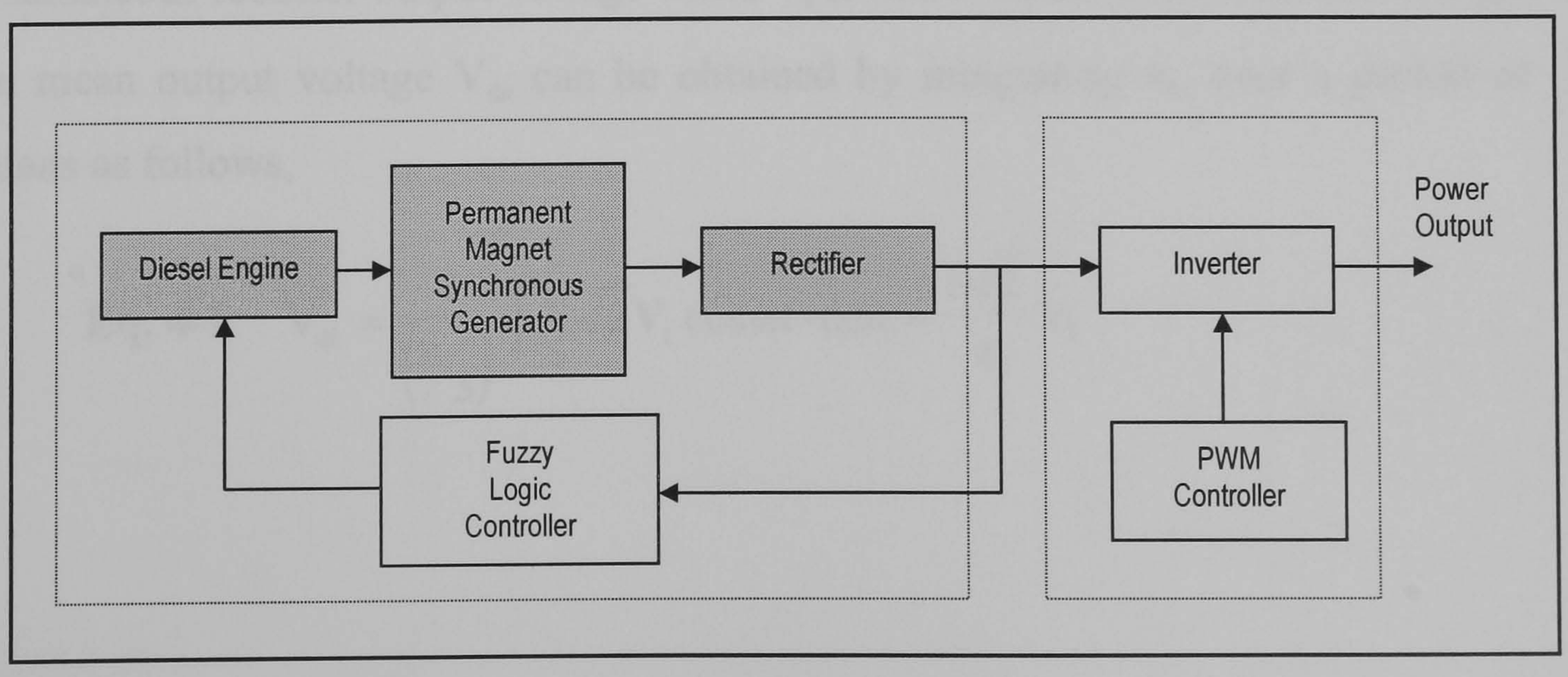


Figure 4-1 *Block diagram of complete system.*

4.1 Rectifier Circuit

A circuit diagram of a three phase uncontrolled (diode) rectifier attached to a Y-connected sinusoidal voltage source is shown in *Figure 4-2*. This circuit is used to convert the a.c. output of the generator terminals into d.c. power. The phases are marked *a*, *b* and *c*. The branches of the rectifier bridge are connected to the generator terminals, therefore the line to line voltage $v_{l\text{-}l}$ of the rectifier is equal to the generator terminal voltage v_t .

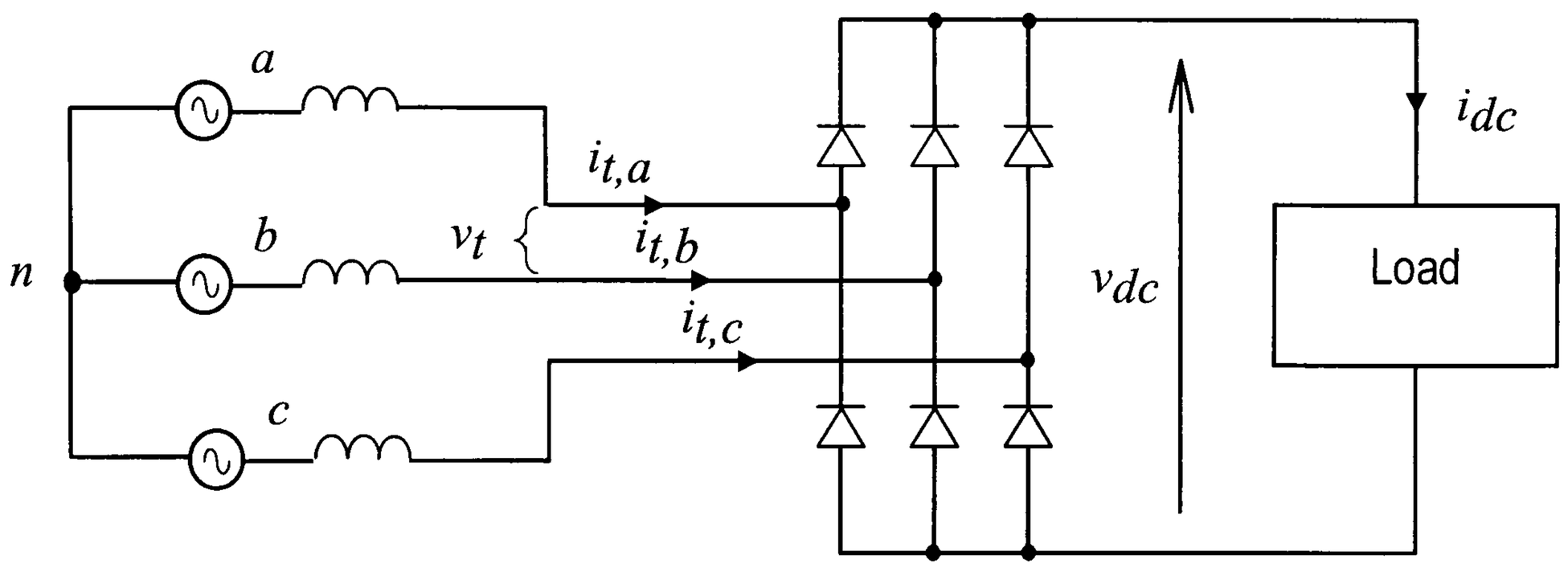


Figure 4-2 *Three phase uncontrolled rectifier.*

4.1.1 Voltage Analysis

Figure 4-3 shows the waveforms of various voltages taken from the circuit in *Figure 4-2*. Waveforms $v_{a\text{-}n}$, $v_{b\text{-}n}$ and $v_{c\text{-}n}$ are the line-neutral voltages of each phase. v_{dc} is the instantaneous rectifier output voltage while V_t is r.m.s. value of the line-line voltage, v_t. The mean output voltage V_{dc} can be obtained by integrating v_{dc} over a period of $\pi/3$ radians as follows,

Eq. 4-1 $$V_{dc} = \frac{1}{(\pi/3)} \int_{\pi/6}^{\pi/6} \sqrt{2} V_t \cos\omega t \cdot d\omega t = \frac{3\sqrt{2}}{\pi} V_t$$

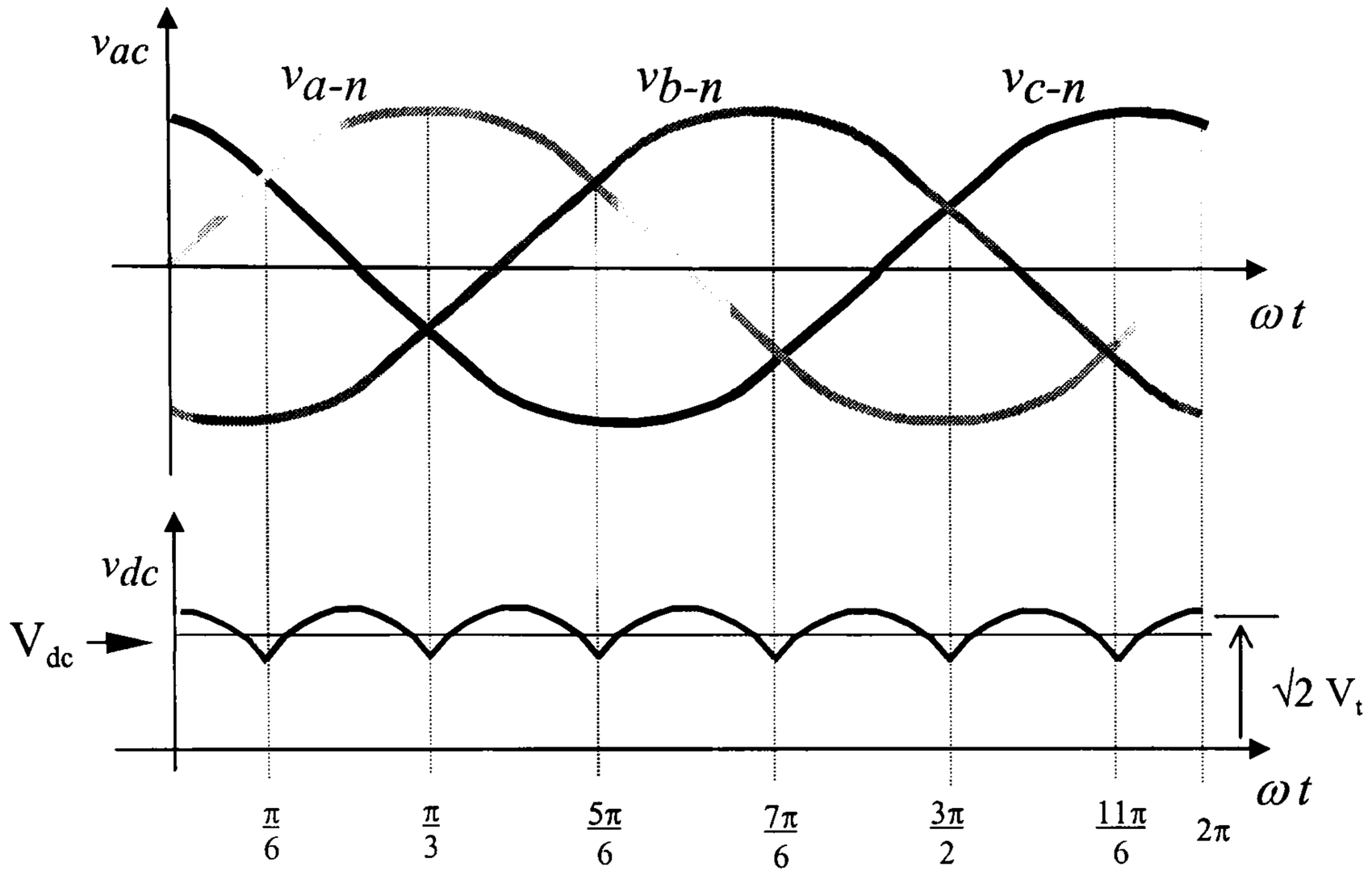


Figure 4-3 *Voltage waveforms of rectifier circuit.*

4.1.2 Current Analysis

The current waveforms of the rectifier circuit are shown in *Figure 4-4*. Because the line current waveform $i_{t,a}$ is not sinusoidal, the r.m.s value cannot be obtained simply with a division by $\sqrt{2}$.

From the definition of *root mean square*, the r.m.s. of $i_{t,a}$ is given by

$$\text{r.m.s. : } I_t = \sqrt{\frac{i_1^{\,2} + i_2^{\,2} + i_3^{\,2} \ldots + i_n^{\,2}}{n}}$$

Taking samples at every $\pi/6$ interval,

$$I_t = \sqrt{\frac{[4 \times (I_{dc})^2] + [4 \times (-I_{dc})^2] + [4 \times 0^2]}{12}}$$

Eq. 4-2 $$I_t = \sqrt{2/3} \cdot I_{dc}$$

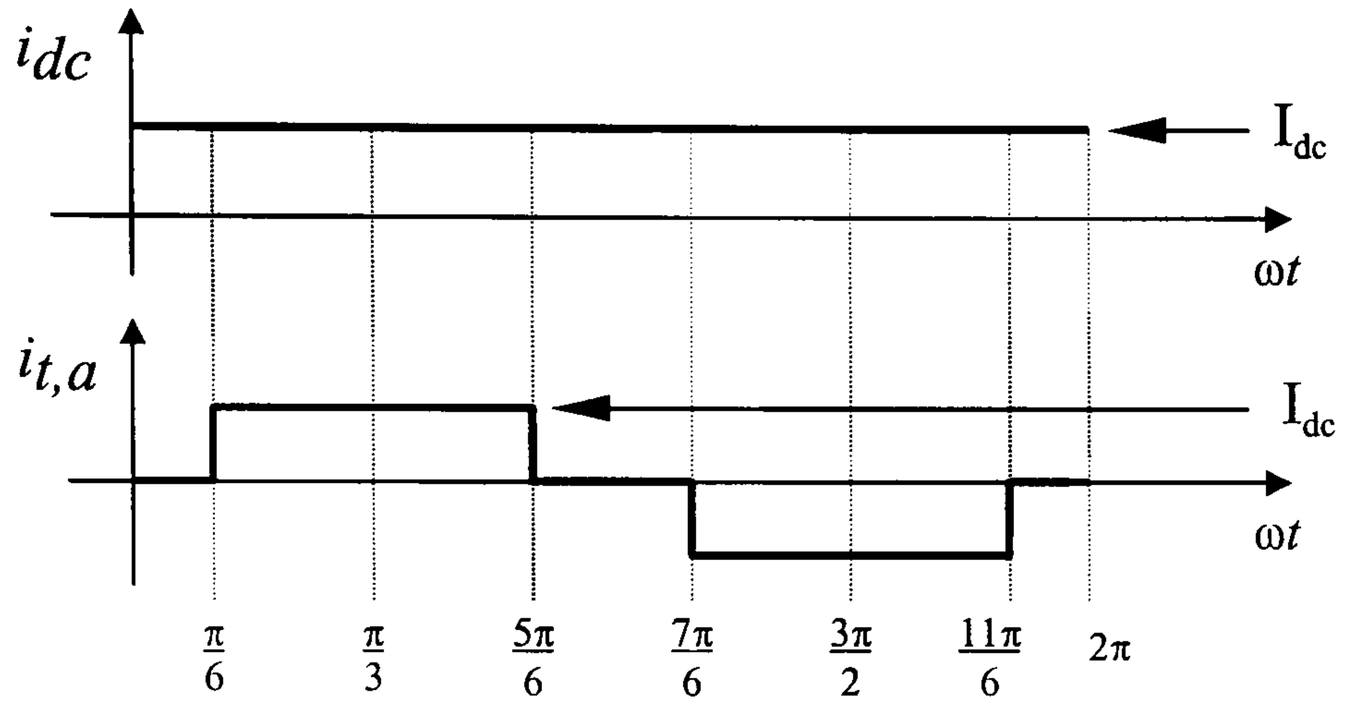


Figure 4-4 *Current waveform of rectifier circuit.*

4.2 Synchronous Generator

4.2.1 Voltage Analysis

The operation of the synchronous generator is based on Faraday's law of electromagnetic induction which states that:

The total electromagnetic force (e.m.f.) generated in a closed circuit is equal to the negative time rate of change of the flux linkages linking the circuit.

This law was derived from Faraday's observation that an e.m.f. is induced in a conductor or circuit placed in a magnetic field when there is a change of flux linkages linking the circuit. The flux changes can be the result of a relative motion between the circuit and the flux or the effect of a varying magnetic flux. Faraday's Law can be represented by the following mathematical equation,

Eq. 4-3 $$e(t) = -\frac{d}{dt}\lambda$$

where *e(t)* is the generated e.m.f. and λ is the flux linkage.

In a synchronous machine, the magnetic flux is provided either by a permanent magnet or an electromagnet which is placed inside a set of windings as shown *Figure 4-5*. The flux linkages linking the winding is made to vary by physically rotating the magnet. In the electrical machine theory, the rotating component is called the rotor and the stationary component is called the stator. The rotor can be a permanent magnet, or an electromagnet with a field winding. Electromagnet generators are easier to control and therefore more common. The stator houses the armature windings. The armature windings are the windings in which the e.m.f. is induced. *Figure 4-5* shows a three phase synchronous machine, with armature windings marked *aa'*, *bb'*, and *cc'*.

If the angular velocity of the rotating field in radians per second is ω, then

Eq. 4-4 $\varphi = \omega t$

Considering only phase *a*, if the rotor moves by an angle φ, the flux linking *aa'* is given by,

Eq. 4-5 $\lambda = N \phi \cos \varphi$

where N is the number of turns in *aa'* and ϕ is the air gap flux.

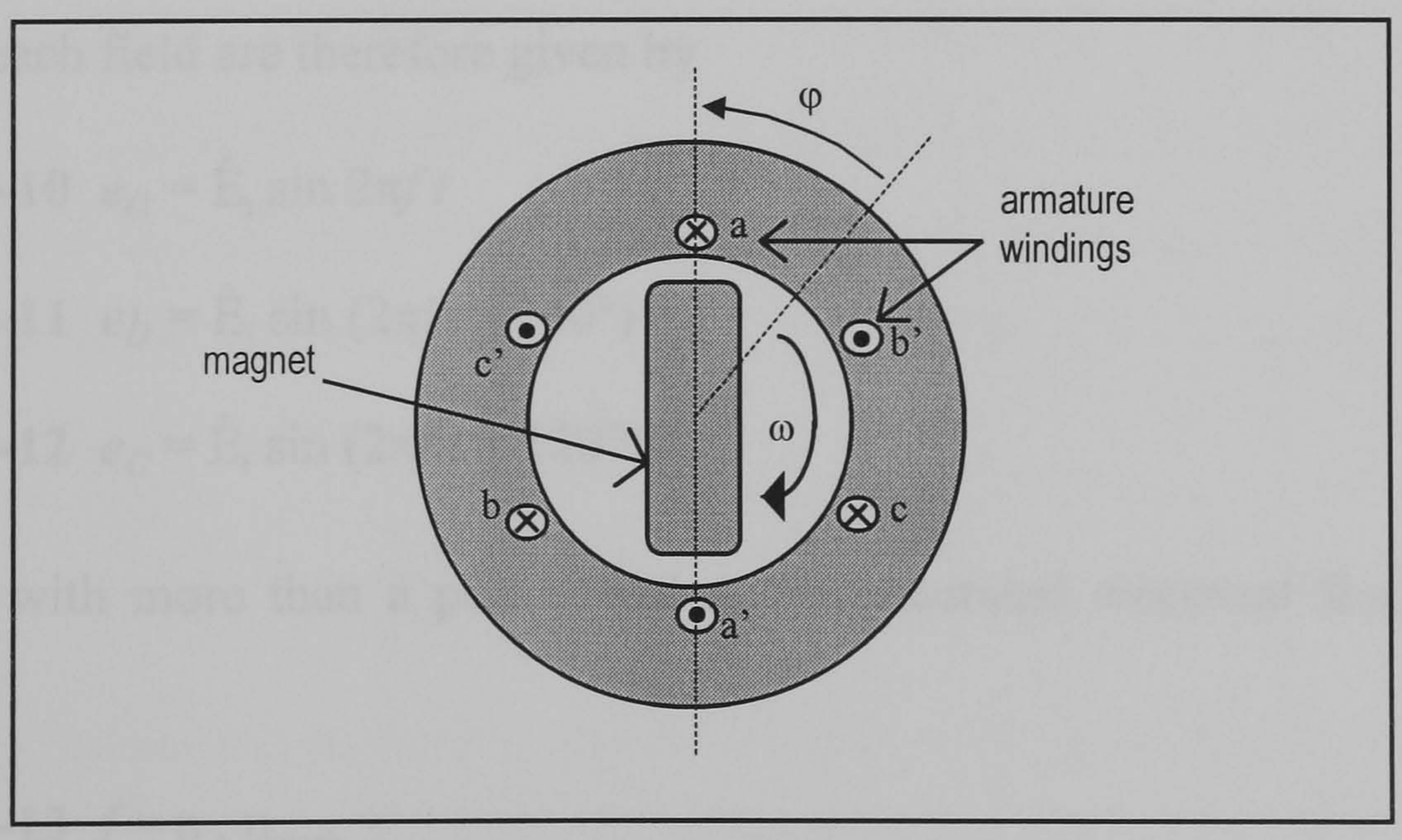


Figure 4-5 *Three phase synchronous machine.*

Therefore, from *Eq. 4-3*, the e.m.f. induced per phase is given by

$$\textbf{Eq. 4-6} \quad e_a = -\frac{d\lambda}{dt} = -\frac{d\lambda}{d\varphi}\cdot\frac{d\varphi}{dt} = (\mathrm{N}\cdot\phi\cdot\sin\varphi)\cdot\frac{d\varphi}{dt}$$

Substituting φ for ωt and $d\varphi/dt$ for $2\pi f$ yields

$$\textbf{Eq. 4-7} \quad e_a = 2\pi f \mathrm{N}\,\phi\,\sin\omega t$$

In practice, the constructions of the machine, such as the pitch and distribution of the coils, assert influence on the waveshape of the induced e.m.f. A reduction factor known as the *winding factor* $\mathrm{K_W}$ has to be applied to account for this.

Therefore, the induced e.m.f. per phase becomes

$$\textbf{Eq. 4-8} \quad e_a = 2\pi f \mathrm{N}\,\phi \mathrm{K_W}\,\sin\omega t$$

Eq. 4-8 can also be written as

$$\textbf{Eq. 4-9} \quad e_a = \hat{\mathrm{E}}_\mathrm{i}\sin 2\pi f t$$

where $\hat{\mathrm{E}}_\mathrm{i}$ is the peak value of the induced e.m.f. and f is its frequency.

Using phase a as reference and because phases a, b and c are displaced by 120°, the voltages for each field are therefore given by

$$\textbf{Eq. 4-10} \quad e_a = \hat{\mathrm{E}}_\mathrm{i}\sin 2\pi f t$$

$$\textbf{Eq. 4-11} \quad e_b = \hat{\mathrm{E}}_\mathrm{i}\sin (2\pi f t - 120°)$$

$$\textbf{Eq. 4-12} \quad e_c = \hat{\mathrm{E}}_\mathrm{i}\sin (2\pi f t + 120°)$$

In machines with more than a pair of poles, the generated electrical frequency, f, is given by,

$$\textbf{Eq. 4-13} \quad f = \mathrm{p}\,.\,\mathrm{n_{mech}}$$

where $\mathrm{n_{mech}}$ is the mechanical synchronous speed in revolutions per second and p is the number of pole-pairs in the machine.

4.2.2 Equivalent Circuit Model

An equivalent circuit model of a synchronous generator can be derived to study the voltage and current characteristics of the machine. The flow of current in the field winding I_f induces a flux ϕ_i in the air gap which links with the armature winding. Similarly, the flow of current in the armature winding also produces a flux, ϕ_a . The armature flux ϕ_a comprises of two components, the *armature reaction flux* ϕ_{ar}, and the *leakage flux* ϕ_{al}. The larger component, ϕ_{ar} , is established in the air gap and links with the field winding, while ϕ_{al} does not exist in the air gap and links only with the armature winding. Therefore, the resultant flux in the air gap ϕ_{ag} is made up of ϕ_i and ϕ_{ar}.

Eq. 4-14 $\phi_{ag} = \phi_i + \phi_{ar}$

ϕ_{ag} induces a voltage in the stator winding, E_{ag} which lags ϕ_{ag} by 90°. According to the Superposition Theorem, E_{ag} comprises two components, induced by the corresponding flux component.

Hence,

Eq. 4-15 $E_{ag} = E_i + E_{ar}$

E_i is the induced e.m.f. also known as the internal generated voltage of the machine. A diagram of the vectors concerned is shown in *Figure 4-6(a)*. It shows the armature reaction voltage E_{ar}, lagging ϕ_{ar} and I_a by 90°. In other words, $-E_{ar}$ leads I_a by 90°. A voltage leading a current by 90° can be represented as the voltage drop across a reactance. Hence $-E_{ar}$ can be represented as the voltage drop across a reactance, say X_{ar}, where X_{ar}, is known as the *magnetising reactance.* Therefore, by substituting $(-E_{ar})$ with $(I_a jX_{ar})$, *Eq. 4-15* can be rewritten as

Eq. 4-16 $E_i = I_a jX_{ar} + E_{ag}$

Figure 4-6(b) shows an equivalent circuit of the model described by *Eq. 4-16*. To include the terminal voltage per phase V_ϕ (instead of just the air gap voltage, E_{ag}), the leakage reactance X_{al} and the effective armature winding resistance R_a have to be taken into account. The effective resistance of R_a is approximately 1.6 times its d.c. resistance as a result of skin effect and operating temperature effects. *Figure 4-6(c)* shows the complete equivalent circuit of a synchronous machine.

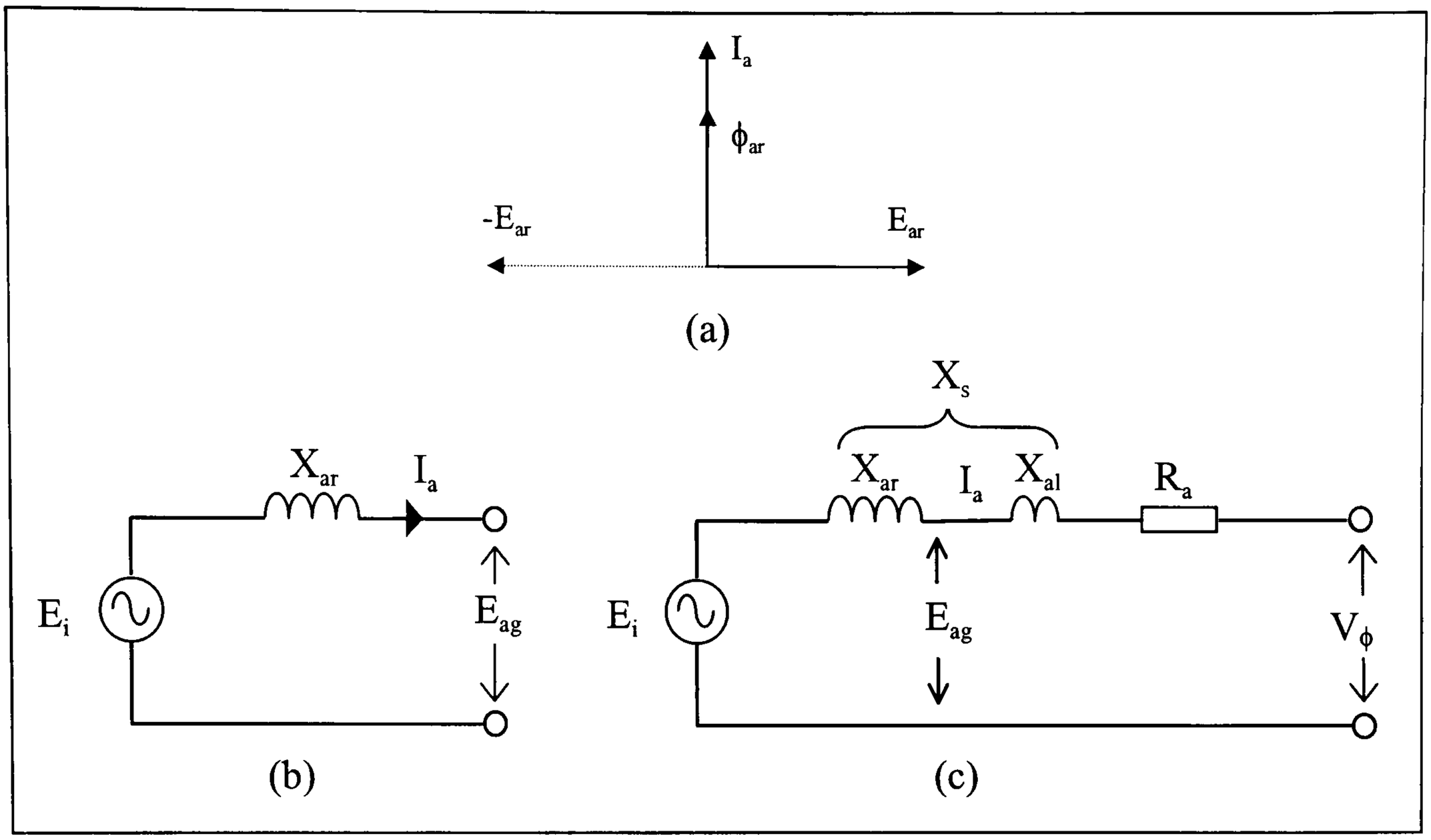


Figure 4-6 *Equivalent circuit model of a synchronous generator.*

X_{ar} and X_{al} can be lumped together as a single reactance denoted by X_s, which is known as the *synchronous reactance*.

Eq. 4-17 $X_s = X_{ar} + X_{al}$

The synchronous impedance is given by

Eq. 4-18 $Z_s = R_a + j\, X_s$

The steady state mathematical model of the synchronous machine can therefore be described by the equation,

Eq. 4-19 $E_i = I_a\,(\,R_a + j\, X_s\,) + V_\phi$

The equivalent circuit model is a steady state model and does not provide any information about the dynamics of the machine. Transient response to load changes, faults and other disturbances cannot be properly analysed using this model. Dynamic models of the synchronous machine can be derived from *The Generalised Theory of Electrical Machines* and are widely used in the design of classical control systems. The design of controllers are derived from the plant model. However, because the control system presented in this thesis is not developed from a model of the plant, the accuracy of the model does not directly influence to the performance of the design. Instead, the

From *Eq. 4-21* and the OCC plot, it can be assumed that flux ϕ is directly proportional to the field current I_f in the linear region. Therefore,

$$\phi \propto I_f$$

Eq. 4-22 $\phi = G \cdot I_f$ where G is the proportionality constant

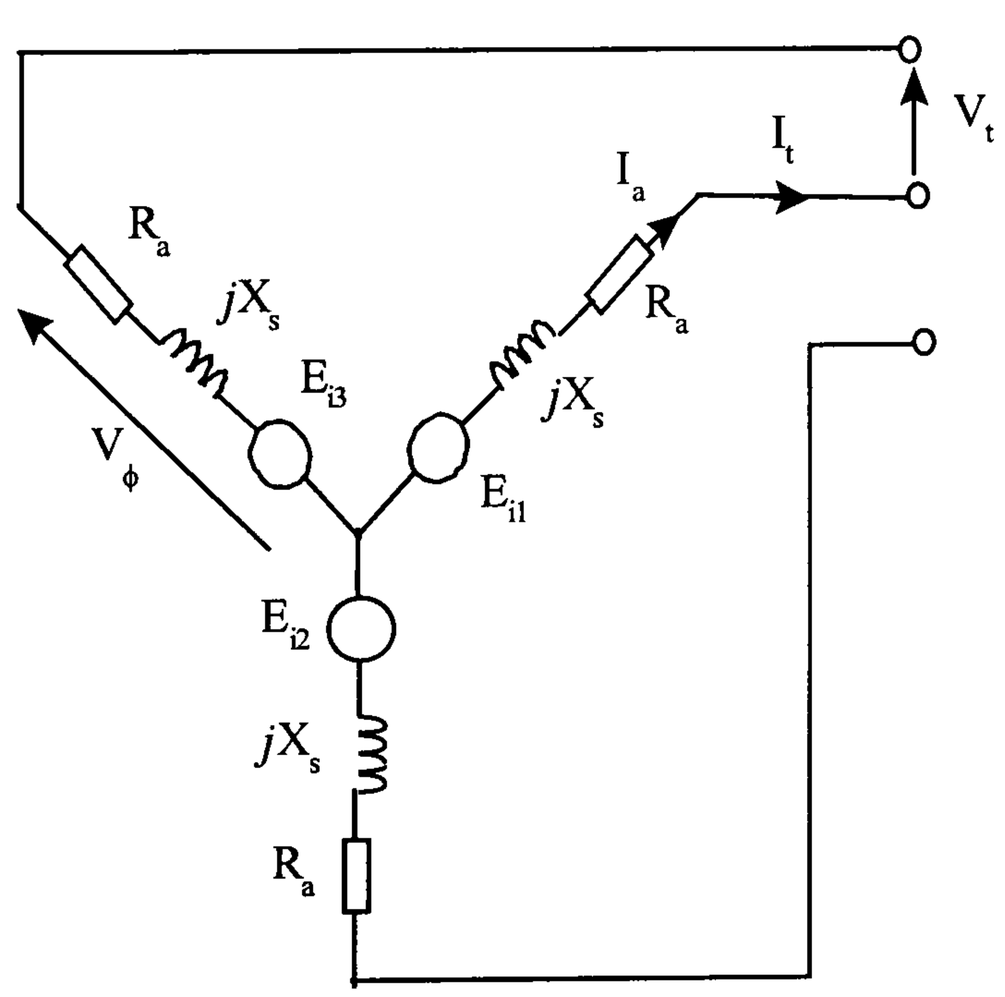
From *Eq. 4-19*, *Eq. 4-21* and *Eq. 4-22*, the synchronous generator can be described by the steady state model,

Eq. 4-23 $E_i = K \cdot G \cdot I_f \cdot n_{mech}$

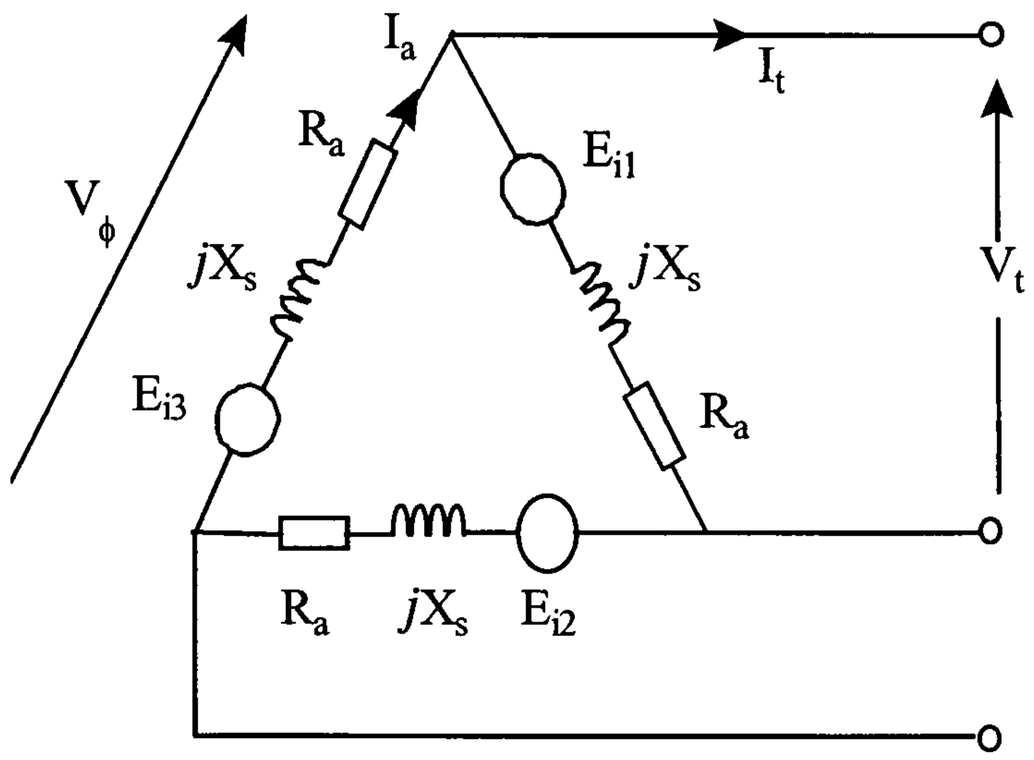
Eq. 4-24 $V_\phi = E_i - I_a (j X_s + R_a)$

where $K = \sqrt{2}\, \pi\, N K_W\, p$.

Eq. 4-23 gives the phase voltage of the generator output. The generator's terminal voltage may or may not be the same as its phase voltage, depending on the configuration of its windings. The three phases of a synchronous generator can either be connected in a star (Y) configuration or delta (Δ) configuration as shown by *Figure 4-8*.



(a)



(b)

Figure 4-8 *Configuration of synchronous generator windings (a) star-connection (b) delta connection.*

model is used to study the response of the design and to verify its functionality through simulations. Although a dynamic model would provide a more comprehensive analysis, the steady state model is sufficient for the task.

4.3 Generator-Rectifier Model

From *Eq.* 4-8 and *Eq. 4-13*, the peak value of the internal generated voltage is found to be

$$\textbf{Eq. 4-20} \quad \hat{E}_i = 2\pi N \phi K_W \, p \, n_{mech}$$

Because the generated voltage is sinusoidal, its r.m.s. value is given by,

$$\textbf{Eq. 4-21} \quad E_i = \sqrt{2}\,\pi \, N \phi K_W \, p \, n_{mech}$$

For an electromagnet generator the flux in the machine is controlled by its field current. The relationship between the flux and the field current can be observed by performing an *open circuit test* on the machine and plotting the results on a 'E_i vs. I_f' graph which is sometimes called the Open Circuit Characteristic (OCC) plot. Details of the test can be found in any text book on Electrical Machines. *Figure 4-7* shows a typical OCC plot of a synchronous generator [58]. The curve is observed to be linear until some saturation starts to occur at high field current. This is due to iron saturation in the synchronous generator.

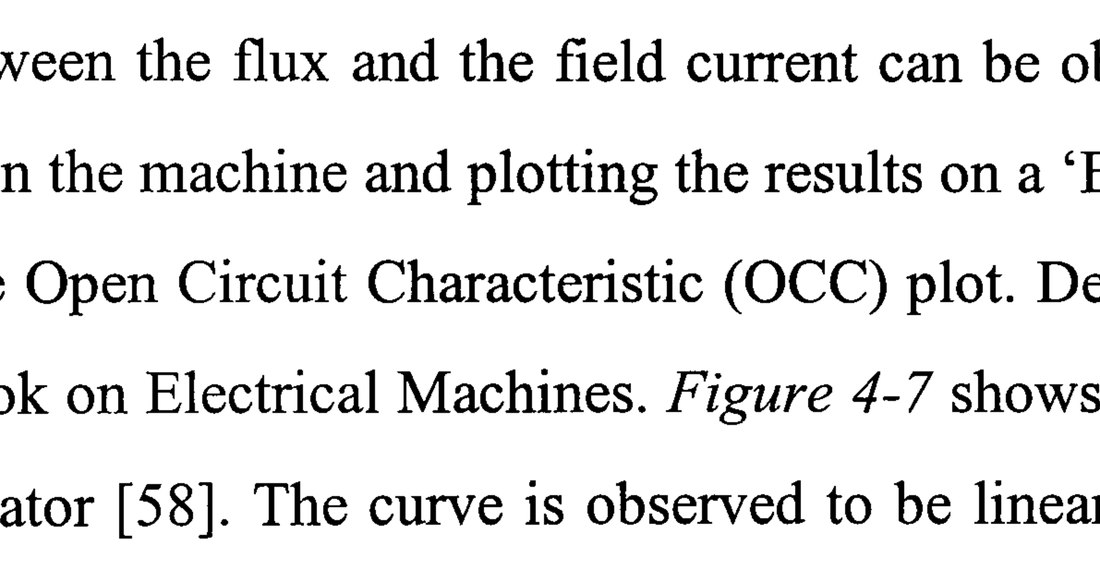


Figure 4-7 *A typical open circuit plot for a synchronous generator.*

From *Eq. 4-21* and the OCC plot, it can be assumed that flux ϕ is directly proportional to the field current I_f in the linear region. Therefore,

$$\phi \propto I_f$$

Eq. 4-22 $\phi = G \cdot I_f$ where G is the proportionality constant

From *Eq. 4-19*, *Eq. 4-21* and *Eq. 4-22*, the synchronous generator can be described by the steady state model,

Eq. 4-23 $E_i = K \cdot G \cdot I_f \cdot n_{mech}$

Eq. 4-24 $V_\phi = E_i - I_a (j X_s + R_a)$

where $K = \sqrt{2}\, \pi\, N K_W\, p$.

Eq. 4-23 gives the phase voltage of the generator output. The generator's terminal voltage may or may not be the same as its phase voltage, depending on the configuration of its windings. The three phases of a synchronous generator can either be connected in a star (Y) configuration or delta (Δ) configuration as shown by *Figure 4-8*.

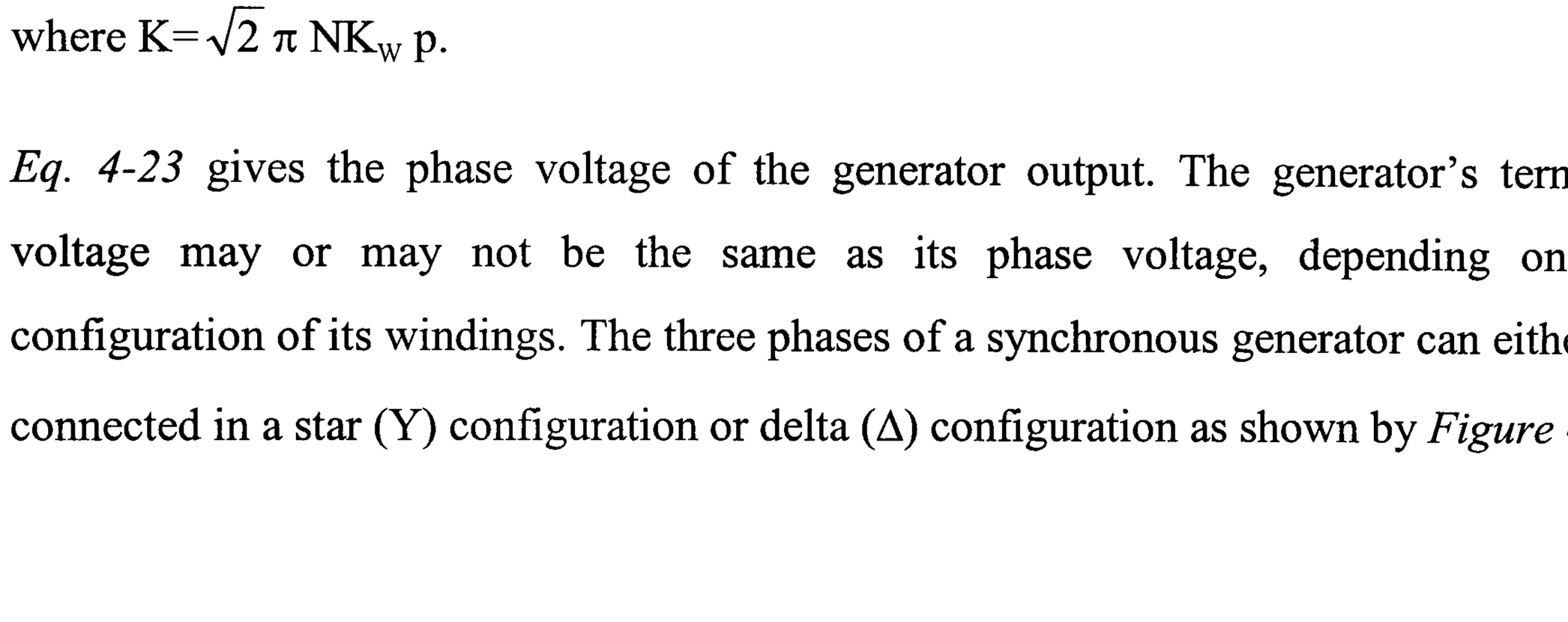


Figure 4-8 *Configuration of synchronous generator windings (a) star-connection (b) delta connection.*

For a Y-connected generator,

Eq. 4-25 $V_t = \sqrt{3}\, V_\phi$ and $I_t = I_a$

For a Δ-connected generator,

Eq. 4-26 $V_t = V_\phi$ and $I_t = \sqrt{3}\, I_a$

where

V_t is the generator's terminal voltage

I_t is the generator's terminal current

V_ϕ is the generator's phase voltage

I_a is the armature current or the generator's phase current.

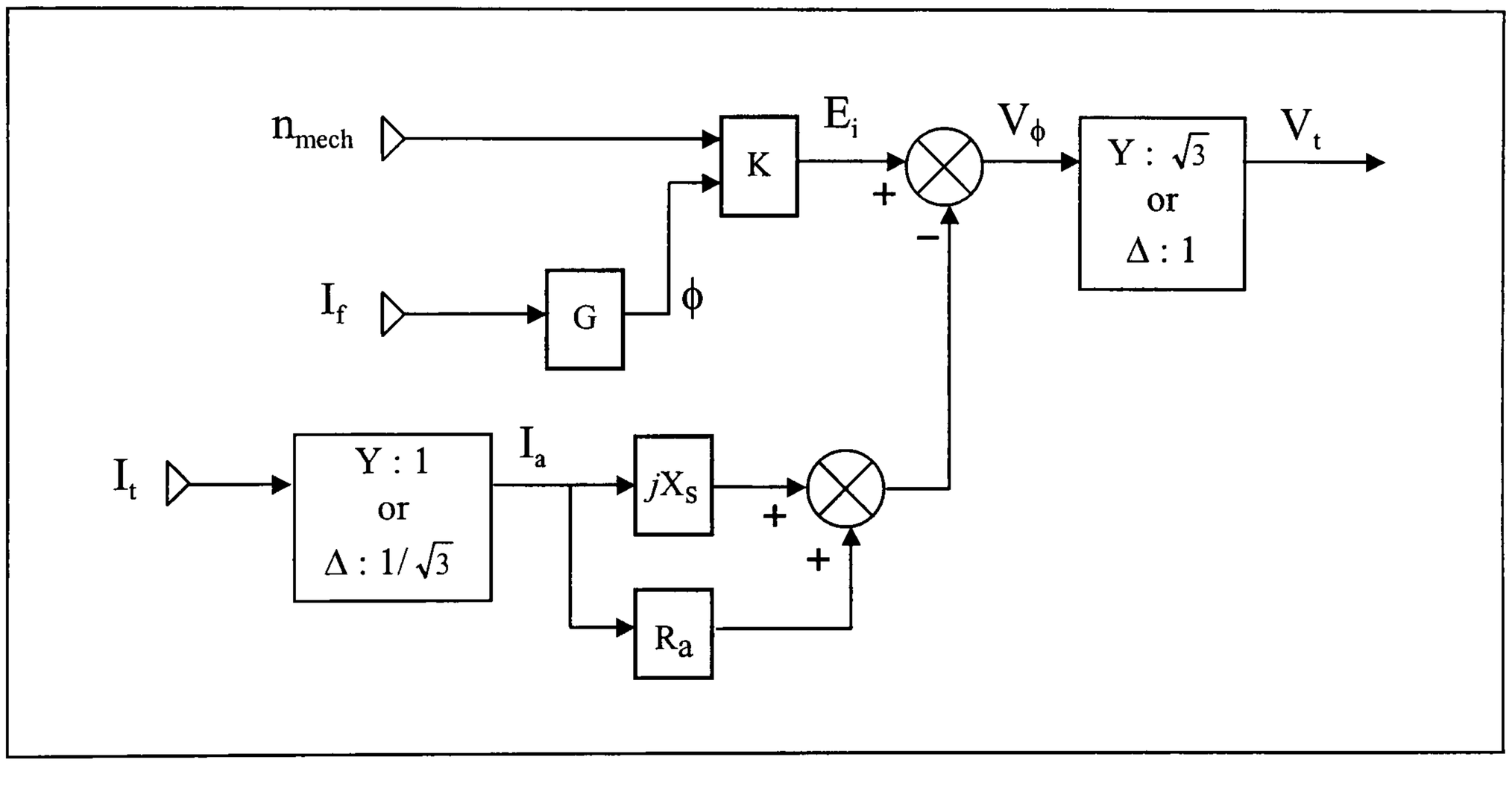


Figure 4-9 *Block diagram based on equivalent circuit model.*

Figure 4-9 shows a block diagram of the synchronous generator model based on the equivalent circuit model. V_t is the complex r.m.s. voltage of the generator terminal. For the present application, this model has to be further developed to include a three phase rectifier and also to consider the torque characteristics as well as the power factor. A phasor diagram of the generator voltages and currents is shown in *Figure 4-10*. The angle between the phase voltage V_ϕ and the internal generated voltage E_i is known as the power angle δ. For generator operation (as opposed to motor operation), the power angle is always positive, as shown in *Figure 4-10*. The diagram also shows a lagging

power factor condition whereby the current I_a lags V_ϕ by an angle θ. The power factor is given by $\cos \theta$.

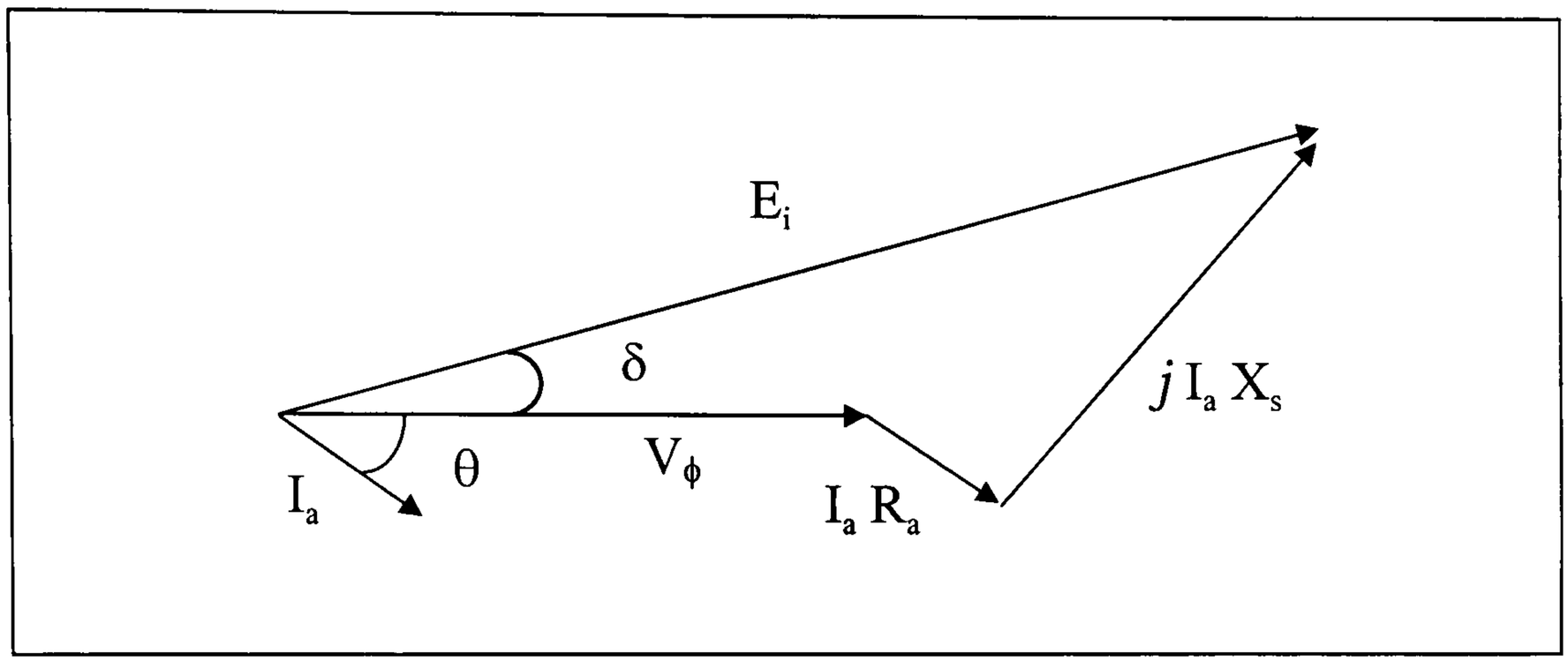


Figure 4-10 *Phasor diagram of a Synchronous generator.*

Using V_ϕ as reference, the equivalent circuit model can be expressed as

$$V_\phi \angle 0° = E_a \angle \delta - I_a \angle \theta \, (R_a + jX_s)$$

$$V_\phi = \{ E_i \cos \delta - I_a R_a \cos \theta + I_a X_s \sin \theta \} + j \{ E_i \sin \delta - I_a X_s \cos \theta - I_a R_a \sin \theta \}$$

Because V_ϕ is the reference, its imaginary component is zero, therefore,

Eq. 4-27 $$V_\phi = E_i \cos \delta - I_a R_a \cos \theta + I_a X_s \sin \theta$$

and

$$E_i \sin \delta - I_a X_s \cos \theta - I_a R_a \sin \theta = 0$$

Eq. 4-28 $$\delta = \sin^{-1}\left[\frac{I_a}{E_i}(X_s \cos\theta + R_a \sin\theta)\right]$$

For a three phase synchronous generator, power is given by

$$P_{real} = 3 \, V_\phi \, I_a \cos \theta$$

Power is the product of torque and speed, hence the induced torque is

$$\tau_{ind} = \frac{3V_\phi I_a}{\omega_{mech}} \cos\theta$$

or

Eq. 4-29
$$\tau_{ind} = \frac{3V_\phi I_a}{2\pi n_{mech}} \cos\theta$$

where ω_{mech} is the mechanical speed in radians per second and n_{mech} is the mechanical speed in revolutions per second.

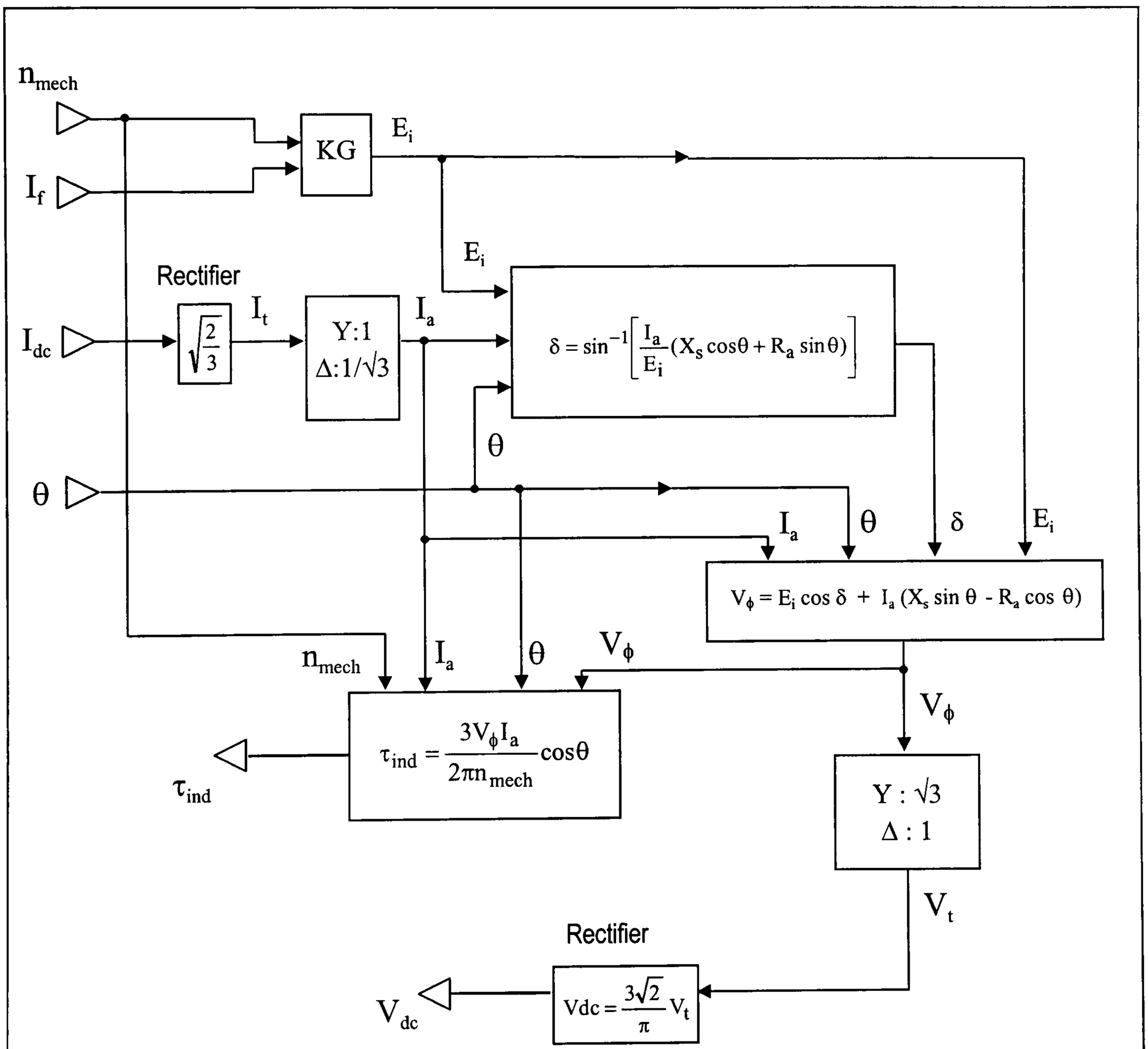


Figure 4-11 *Model of Generator-Rectifier system.*

Figure 4-11 shows the model which is developed to represent the synchronous generator and rectifier. The inputs of this model are the rotational speed n_{mech} , field current I_f , power factor angle θ and the current at the d.c. link I_{dc}, used as a measure of the electrical load. The outputs are the d.c. link voltage V_{dc} and the induced torque τ_{ind} which loads the engine.

To account for the generator winding configuration (Y or Δ), conversions between the phase values and terminal values of voltage and current are based on *Eq. 4-25* and *Eq. 4-26*. The generator phase voltage is obtained from *Eq. 4-27* and *Eq. 4-28* while the induced torque from *Eq. 4-29*.

4.4 Diesel Engine

The second model to be developed is one for the diesel engine. The operation of diesel engines involves a large number of complex processes, with many delays, lags and non-linearities. It is therefore difficult to develop a comprehensive mathematical model to study and analyse the diesel engine. Even manufacturers of control systems and speed governors for diesel engines are known to focus on empirical methods rather than theoretical modelling for analysis. In this thesis, a model of a diesel engine is required to provide a complete representation of the control plant. The aim of this exercise is to develop a 'good enough' model for the functional verification of the control system. Since detailed studies are not necessary at this stage, an approach which favours reduced complexity and incorporates a certain degree of approximation is preferred.

For the purpose of speed control studies, the diesel engine can be represented as a combustion system and inertia as shown by the block diagram in *Figure 4-12*. The combustion system produces an engine torque τ_E as a function of the fuel rack position *y*. The engine torque is used to oppose the load torque τ_L and the difference $\Delta\tau$ is converted into acceleration/deceleration Δn_{mech} in the lumped inertia (engine inertia + load interia).

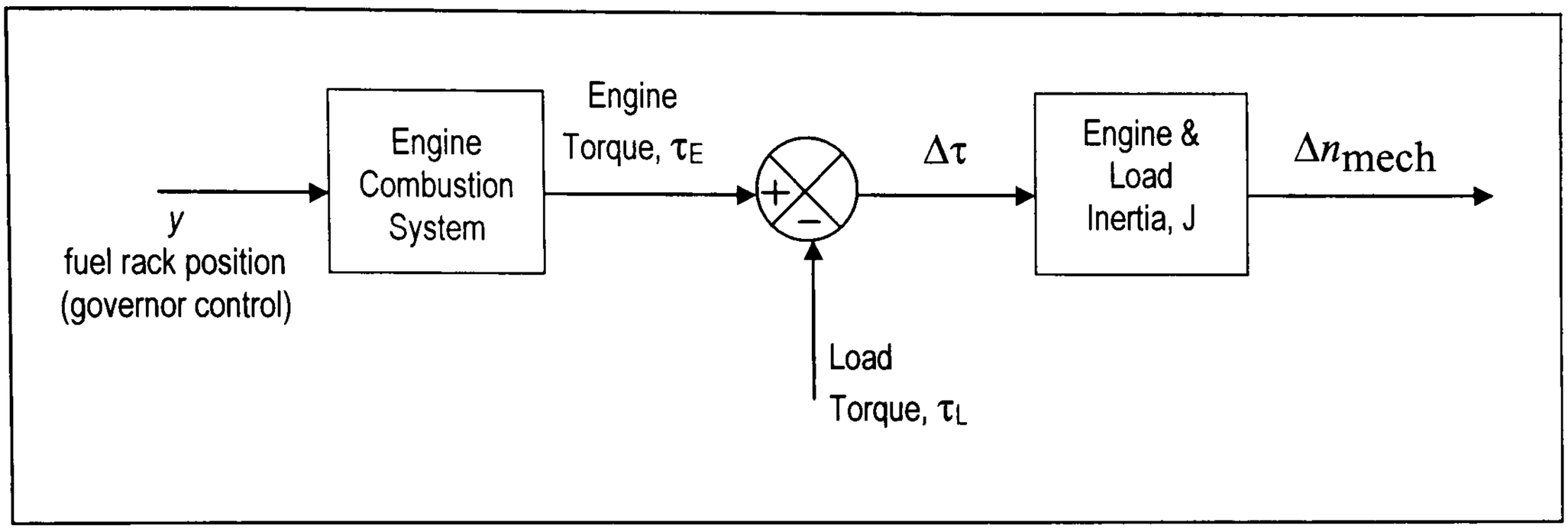


Figure 4-12 *Diesel engine & load representation.*

When the speed of the engine changes, it moves along the engine's torque curve as shown in *Figure 4-13*. Each engine's torque curve is specific to a fuelling rate q. For a given speed, the developed engine torque τ_E can be increased by increasing q. An equilibrium point is reached when the engine's torque curve intersects the load's torque curve.

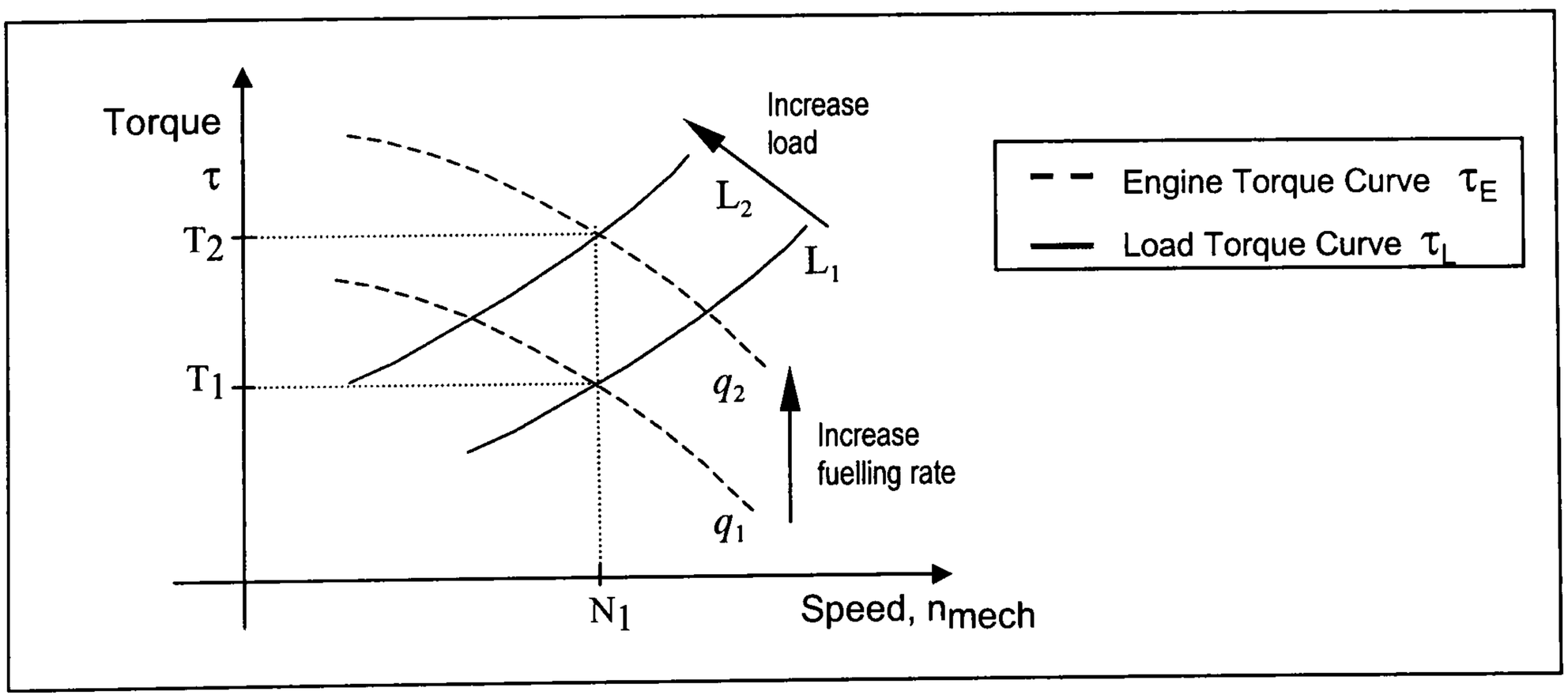


Figure 4-13 *Torque-speed characteristics.*

If the engine torque is written as

Eq. 4-30 $\tau_E = f(n_{mech}, q)$

and the equilibrium point for a given operating condition is when

$$n_{mech} = \tilde{n}_{mech} \quad ; \quad q = \tilde{q}$$

then, expanding *Eq. 4-30* into a Taylor series about the equilibrium point yields

$$\tau_E = f(\tilde{n}_{\text{mech}}, \tilde{q}) + \left[\frac{\partial f}{\partial n_{\text{mech}}}(n_{mech} - \tilde{n}_{\text{mech}}) + \frac{\partial f}{\partial q}(q - \tilde{q}) \right]$$
$$+ \frac{1}{2!}\left[\frac{\partial^2 f}{\partial n_{\text{mech}}^{\ 2}}(n_{mech} - \tilde{n}_{\text{mech}})^2 + \frac{\partial^2 f}{\partial n_{\text{mech}} \partial q}(n_{mech} - \tilde{n}_{\text{mech}})(q - \tilde{q}) \right.$$
$$\left. + \frac{\partial^2 f}{\partial q^2}(q - \tilde{q})^2 \right] + \ldots.$$

If the operating condition is within the vicinity of the equilibrium point, it can be assumed that the higher order terms are negligible, therefore,

$$\tau_E - f(\tilde{n}_{\text{mech}}, \tilde{q}) = \left(\frac{\partial f}{\partial n_{\text{mech}}} \right)(n_{mech} - \tilde{n}_{\text{mech}}) + \left(\frac{\partial f}{\partial q} \right)(q - \tilde{q})$$

Eq. 4-31 $$\tau_E - \tilde{\tau}_E = \left(\frac{\partial f}{\partial n_{\text{mech}}} \right)(n_{mech} - \tilde{n}_{\text{mech}}) + \left(\frac{\partial f}{\partial q} \right)(q - \tilde{q})$$

Using ‘δ’ to denote small changes about the equilibrium point, *Eq.* 4-31 can be written as

Eq. 4-32 $$\delta\tau_E = \left(\frac{\partial \tau_E}{\partial n_{\text{mech}}} \right)_q \delta n_{\text{mech}} + \left(\frac{\partial \tau_E}{\partial q} \right)_{n_{mech}} \delta q$$

At this juncture, a bold step is taken to further simplify the situation by extending the use of *Eq. 4-32* over the entire range of analysis. It is acknowledged that *Eq. 4-32* is a linearised approximation which is only valid within proximity of the operational point from which the equation is derived. However, this linear model is assumed to be capable of providing, at least a qualitative ‘feel’ of the engine’s torque-speed characteristics, even if the quantitative results are only rough approximations. It has to be pointed out that these models are not developed to be used as a starting point for the design of the control system. The controllers presented in this thesis are model-free designs, therefore

there is no danger of the limitations of the models being translated into the control system.

Writing

$$\left(\frac{\partial \tau_E}{\partial n_{mech}}\right)_q = A \qquad \text{and} \qquad \left(\frac{\partial \tau_E}{\partial q}\right)_{n_{mech}} \delta q = B\delta y$$

where δy is the change in fuel rack position y, *Eq. 4-32* can be written as

$$\delta\tau_E = A \cdot \delta n_{mech} + B \cdot \delta y$$

By extending the linear properties over the entire range of analysis, it can be assumed that A and B are constants. Therefore, from the *Superposition Principle*, the engine's torque curve can be represented by the linear equation:

Eq. 4-33 $$\tau_E = A \cdot n_{mech} + B \cdot y$$

The net accelerating torque is given by the difference between the engine's torque and the load torque:

$$\Delta\tau = \tau_E - \tau_L$$

Since

$$\Delta\tau = J\frac{d\omega_m}{dt}$$

where

J is the combined moment of inertia of the engine and generator

ω_m is the mechanical speed in radians per second,

it can be shown that the discrete equivalent can be written as,

$$\frac{C(n_{mech} - n_{mech} \cdot z^{-1})}{T} = \frac{\Delta\tau}{J}$$

Eq. 4-34 $$\Delta n_{mech} = \frac{\Delta\tau}{J} D$$

where

C is a constant

T is the sampling period

D = (T/C)

n_{mech} is the mechanical speed in revolutions per second.

A block diagram of a discrete engine model based on *Eq. 4-33* and *Eq. 4-34* is shown in *Figure 4-14*.

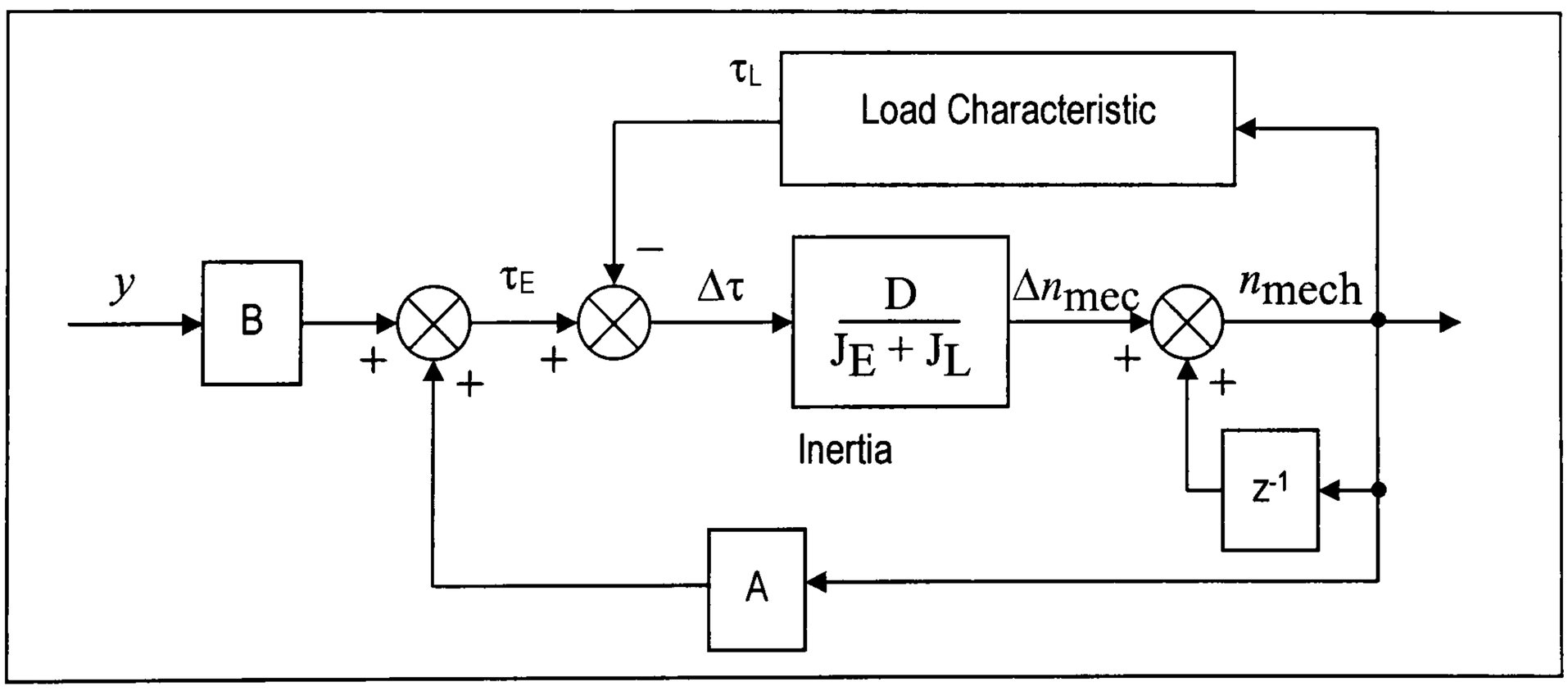


Figure 4-14 *Linearised discrete model of the diesel engine.*

4.5 VHDL Modelling

Although VHDL is a hardware description language and as such, is used primarily for circuit design, it has the basic properties of any software programming language. It is therefore capable of implementing mathematical models. Two VHDL components are designed to represent the models of the generator-rectifier and the diesel engine developed in this chapter. The complete source codes of the models can be found in *Appendix A*. Since the models are designed for the purpose of computer simulations (refer to *Section 6.6*) and not for hardware implementation, it is possible to declare the signals and variables as the type REAL. This gives the design a greater freedom in numerical analysis. It is also possible to incorporate complex mathematical functions such as trigonometry in the design.

The VHDL design of the generator and rectifier model is configured with the entity name **Genrect**. It is designed as a synchronous circuit with five input signals (including **CLK**). Three output signals provide simulated information on the generator's phase voltage **Vph** and induced torque **TG** as well as the d.c. voltage at the rectifier's output **Vdc**. As already touched upon earlier in this chapter, the voltages at the generator's terminals are largely determined by the nature of its winding configuration. Two constants, **YD_CURR** and **YD_VOLT**, which have values defined in the declaration section of the design architecture sets the configuration of the model. In this, they are defined with values that set the model as a Y-connected generator.

The VHDL model for the diesel engine is configured with the entity name **Engine**. The input signals of the model are load torque **TL**, control signal to the fuel actuator **Y**, the clock signal **CLK** and the sampling period **PRD** while the engine's torque **TE** and speed **NE** make up the output signals. It implements the mathematical equations of the engine and, like **Genrect**, the section of the code that does this is self-explanatory. There is, however, a slight addition to the model. To prevent the calculated output values from 'running away' in the event of simulation errors, the output signals are designed to operate within a fixed boundary of values. The maximum and minimum limits of these values are set within the code.

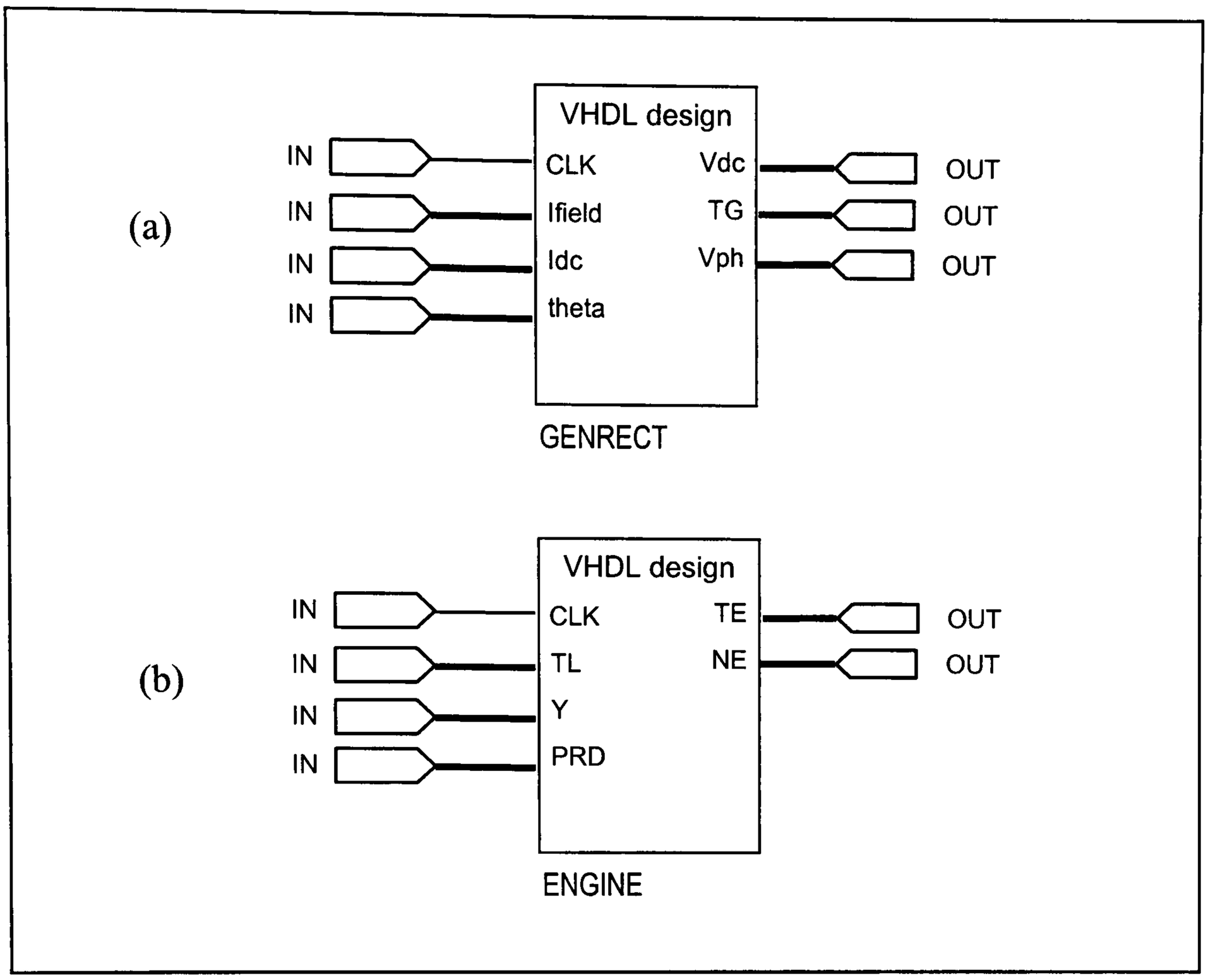
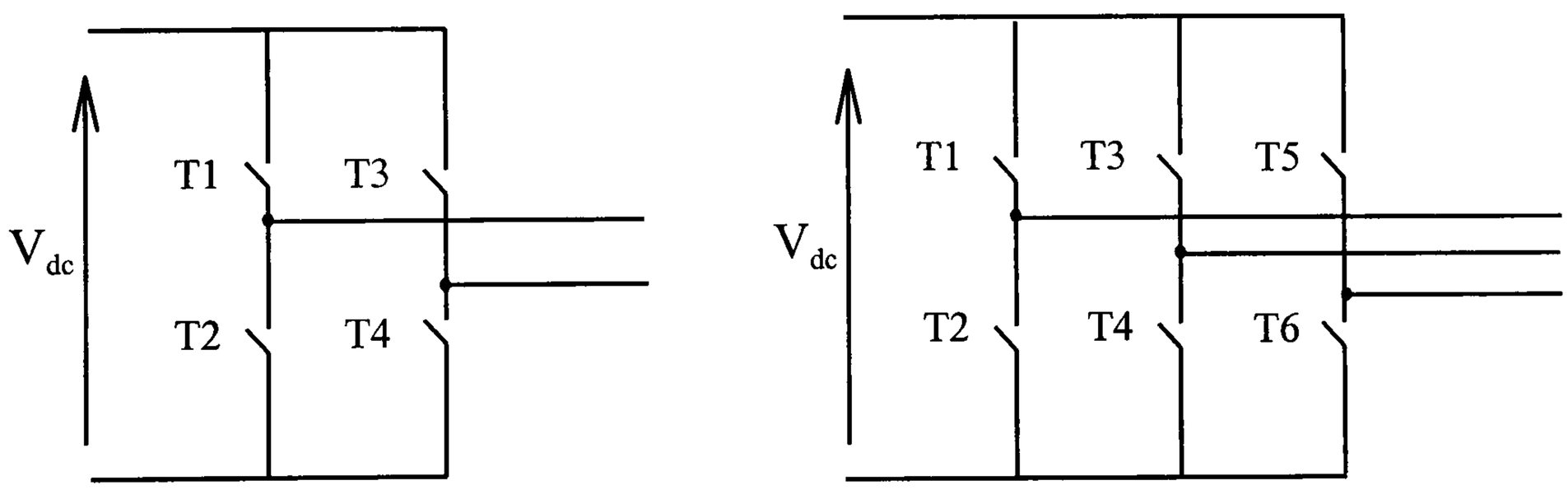


Figure 4-15 *Block diagram of VHDL models (a) Genrect (b) Engine.*

Figure 4-15(a) and *(b)* shows the block diagram of **Genrect** and **Engine** respectively. The two components have their respective input and output ports and can be connected directly to the VHDL designs of the control system for simulation purposes. Further discussion on the use of the plant models in conjunction with the control system in a functional simulation is presented in *Chapter 6*, following a description of the control system designs. Prior to this, the analysis of the second part of the plant, the PWM inverter, is presented in the following chapter. It includes some functional and performance simulations as well as a description of the inverter control system designs.

The diagrams in *Figure 5-2* show the configurations of a single phase and a three phase switch-mode inverters. They are made up of a number of power switches (four for single phase and six for three phase) arranged in a bridge configuration. At present, IGBTs are the most widely used type of power switches for inverter applications due to their controllability and relatively high voltage ratings. Various control strategies can be used to control the power switches such that a sinusoidal waveform can be obtained at the output. However, the raw output voltage usually contains other high frequency harmonic components which have to be eliminated before the desired voltage waveform can be recovered. This is normally achieved using a low pass filter. The amount of harmonic distortion at the output depends largely on the type of switching pattern. Examples of commonly used switching patterns are square wave, quasi square wave and Pulse Width Modulation (PWM). The switching strategy used in this project is Pulse Width Modulation, a technique which produces waveforms which tend to contain less of the low order harmonics than other switching arrangements.



(a) (b)

Figure 5-2 *Switch-mode inverter (a) single phase, (b) three phase.*

5.1 Pulse Width Modulation

Pulse Width Modulation (PWM) is currently the most widely used technique of inverter control and has received considerable attention in the last two decades. The PWM switching scheme essentially involves the strategic variation of the ON and OFF timing periods of each pair of switches in the inverter. This produces a PWM waveform that contains a series of pulses which have the same voltage level but different widths, as shown by the waveform in *Figure 5-3*.

Chapter 5

PWM Inverter Design and Analysis

While the previous chapter discussed the mathematical representation of the diesel engine, synchronous generator and rectifier, this chapter presents an analysis on the second part of the control plant, namely the power inverter system shown as the shaded blocks in *Figure 5-1*. It is well known that a common application of power inverters is in a.c.-d.c.-a.c. systems such as in the variable speed control of a.c. motors, uninterruptible power supply systems and variable speed wind energy conversion systems. In this project, an inverter is used to convert the d.c. voltage at the d.c. link into a constant 50 Hz sinusoidal output voltage.

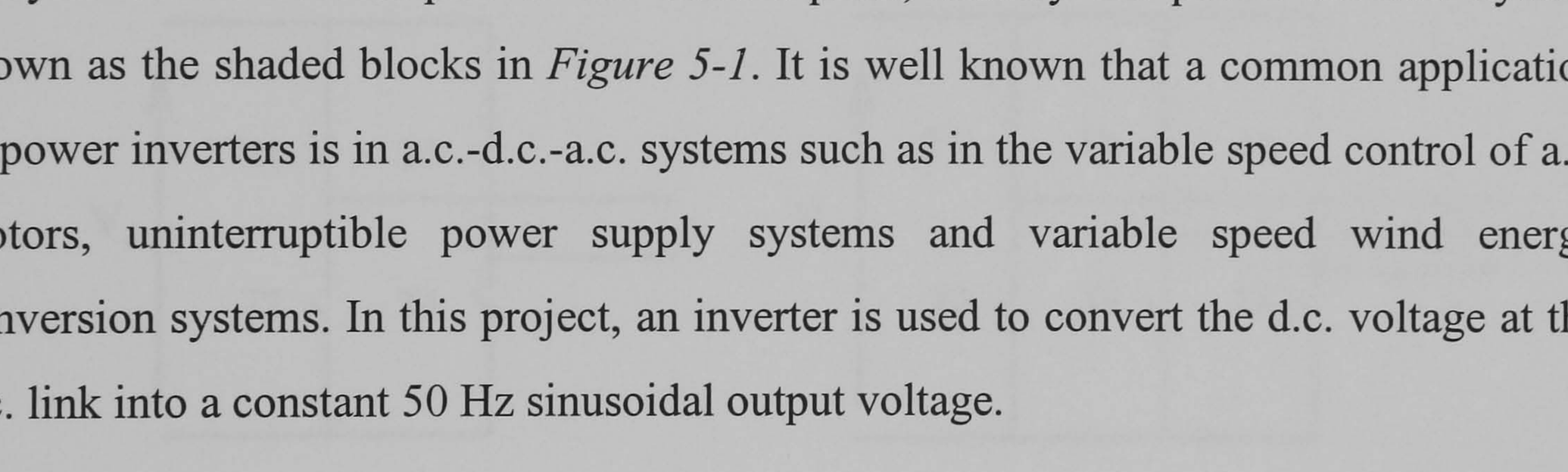
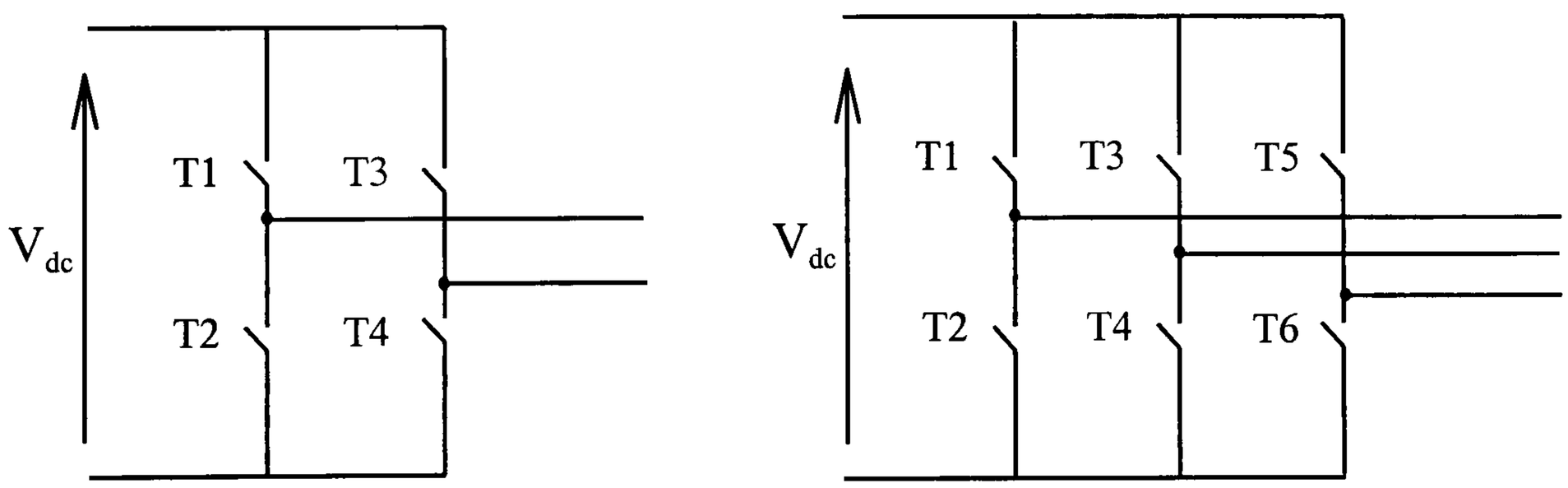


Figure 5-1 *Block diagram of the complete system.*

The diagrams in *Figure 5-2* show the configurations of a single phase and a three phase switch-mode inverters. They are made up of a number of power switches (four for single phase and six for three phase) arranged in a bridge configuration. At present, IGBTs are the most widely used type of power switches for inverter applications due to their controllability and relatively high voltage ratings. Various control strategies can be used to control the power switches such that a sinusoidal waveform can be obtained at the output. However, the raw output voltage usually contains other high frequency harmonic components which have to be eliminated before the desired voltage waveform can be recovered. This is normally achieved using a low pass filter. The amount of harmonic distortion at the output depends largely on the type of switching pattern. Examples of commonly used switching patterns are square wave, quasi square wave and Pulse Width Modulation (PWM). The switching strategy used in this project is Pulse Width Modulation, a technique which produces waveforms which tend to contain less of the low order harmonics than other switching arrangements.



(a) (b)

Figure 5-2 *Switch-mode inverter (a) single phase, (b) three phase.*

5.1 Pulse Width Modulation

Pulse Width Modulation (PWM) is currently the most widely used technique of inverter control and has received considerable attention in the last two decades. The PWM switching scheme essentially involves the strategic variation of the ON and OFF timing periods of each pair of switches in the inverter. This produces a PWM waveform that contains a series of pulses which have the same voltage level but different widths, as shown by the waveform in *Figure 5-3*.

The fundamental component of the PWM switching pattern V_{PWM} in *Figure 5-3* is a sinewave, which is the required output voltage. To obtain the switching pattern, a sinusoidal signal $V_{control}$ is compared with a high frequency triangular carrier wave V_{tri}. This form of PWM control is sometimes called *sinusoidal-PWM* in order to explicitly differentiate it from other forms of PWM control schemes. *Table 5-1* illustrates how $V_{control}$ and V_{tri} can be used to determine the switching pattern in a single phase inverter. The two devices on the same branch (T1&T2 ; T3&T4) must not be ON at the same time, otherwise a short circuit will occur.

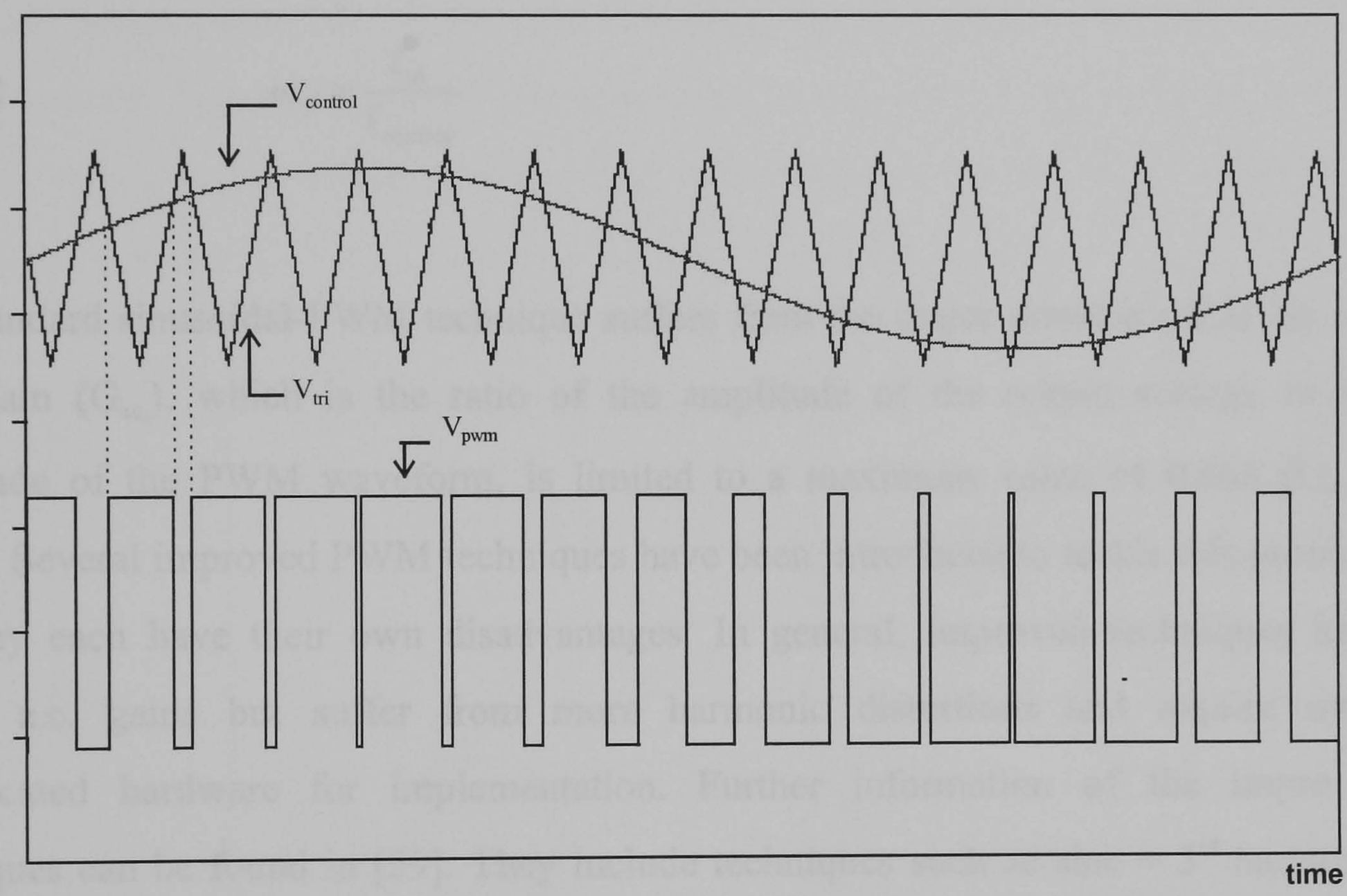


Figure 5-3 *Pulse Width Modulation.*

Table 5-1 *PWM control.*

	T1	T2	T3	T4
$V_{control} \geq V_{tri}$	ON	OFF	OFF	ON
$V_{control} < V_{tri}$	OFF	ON	ON	OFF

In sinusoidal-PWM control schemes, there are two characteristic ratios which are important factors in the design of the controllers.

The amplitude modulation ratio m_a is defined as the ratio of the peak amplitude of the control signal ($\hat{V}_{control}$) to the peak amplitude of the triangular carrier signal ($\hat{V}_{tri}$):

Eq. 5-1
$$m_a = \frac{\hat{V}_{control}}{\hat{V}_{tri}}$$

The frequency modulation ratio mf is defined as the ratio of the triangular carrier frequency (f_{tri}) to the ratio of the control signal frequency ($f_{control}$):

Eq. 5-2
$$m_f = \frac{f_{tri}}{f_{control}}$$

The standard sinusoidal-PWM technique suffers from the major drawback that the a.c. term gain ($G_{ac,}$), which is the ratio of the amplitude of the output voltage to the amplitude of the PWM waveform, is limited to a maximum value of 0.866 ($G_{ac} \leq 0.866$). Several improved PWM techniques have been introduced to tackle this problem but they each have their own disadvantages. In general, improved techniques have higher a.c. gains but suffer from more harmonic distortions and require more complicated hardware for implementation. Further information of the improved techniques can be found in [59]. They include techniques such as sine + 3rd harmonic PWM, harmonic injection and programmed harmonic elimination. Other PWM techniques include Random PWM Schemes and Sliding Mode Control. Random PWM schemes [60, 61], are based on the use of random number generation. They offer a more evenly spread harmonic spectrum and are found to have reduced radio interference, noise and vibration effects. Sliding Mode Control, on the other hand, is described by Jung & Tzou [62] to be especially suitable for closed loop control of power converting systems under load variations. However, improved PWM techniques require a more complex hardware implementation. For the present work, the standard PWM technique is found to be suitable for the application while being easier to implement in hardware compared to the other techniques.

5.2 Design and Simulation

A PWM inverter based on the standard sinewave technique was designed and simulated using TCAD. The switching waveform is obtained by comparing a control sinewave with a triangular wave. *Figure 5-4* shows the schematic diagram of a single phase inverter in TCADSchem. From the diagram, it is obvious that if T1 and T2 are both ON at the same time, a short circuit fault, sometimes called a 'shoot through' will occur. In the previous section, it was assumed that the switches are ideal, and that T1 switches ON at the exact moment when T2 switches OFF and vice versa. However, in practice, switching devices such as IGBTs and Mosfets cannot be switched ON or OFF instantaneously. Therefore, precautions have to be taken such that 'shoot through' does not occur as a result of the switching delays and finite turn on/off times. This can be achieved by introducing a blanking time T_Δ into the PWM waveforms.

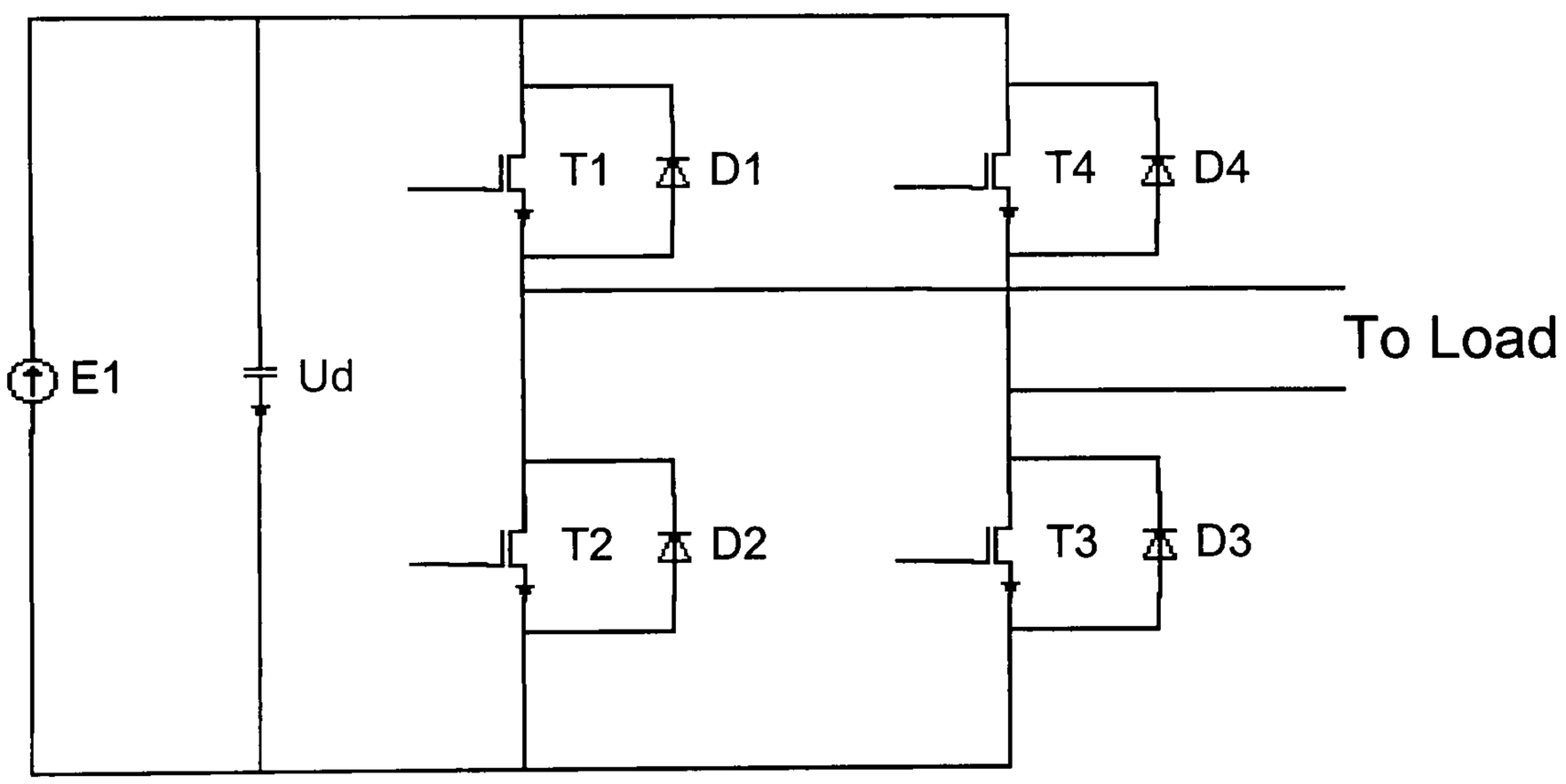


Figure 5-4 *PWM Inverter.*

5.2.1 Blanking Time

Figure 5-5 shows the effects of blanking time on the PWM switching patterns. In the ideal case, T1 and T2 change states at the exact moment. However in practice, a short blanking time T_Δ is introduced , during which both devices are supposed to be in the OFF state. The duration of T_Δ will depend on the maximum switching delay and turn ON/OFF times of the device.

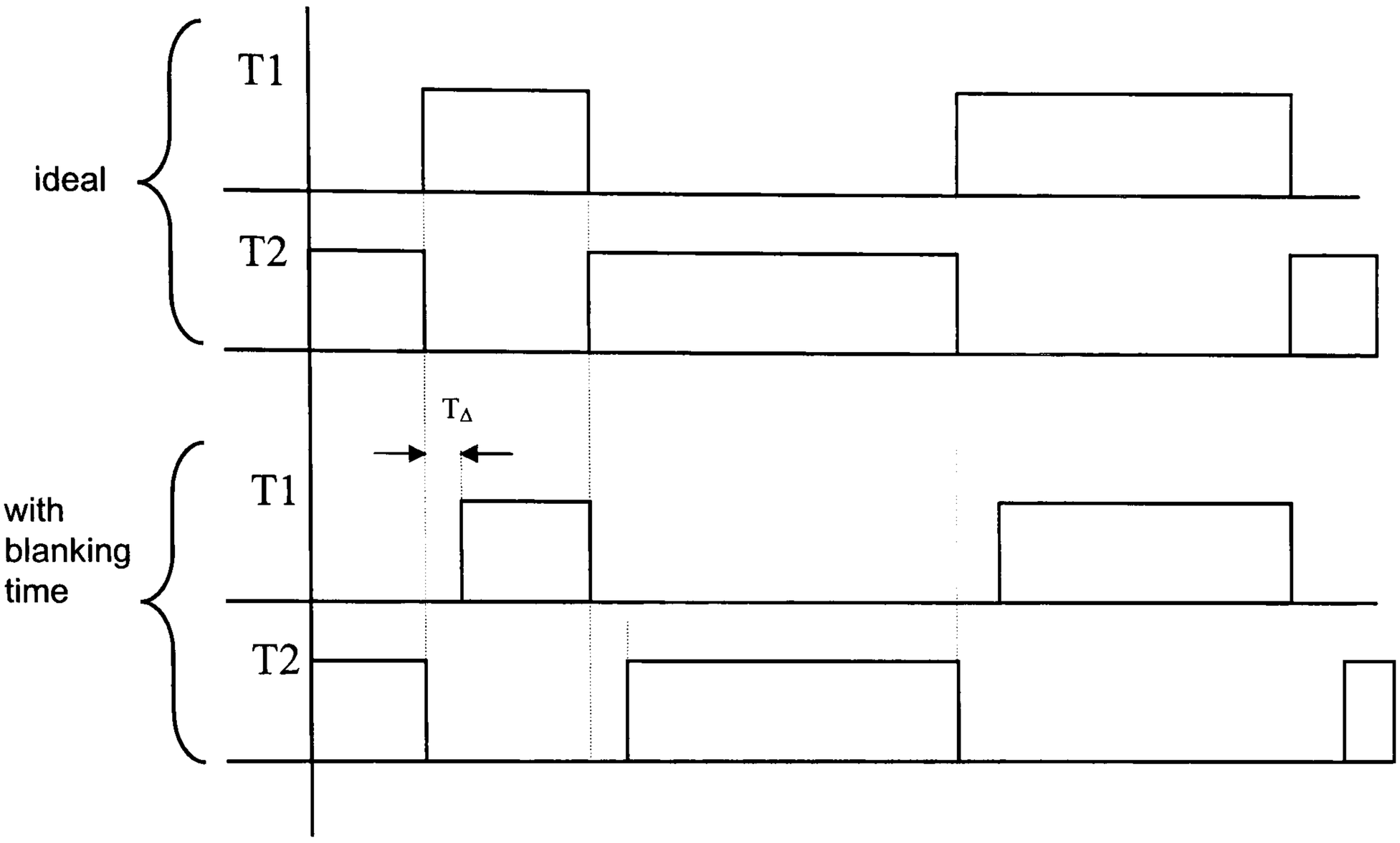


Figure 5-5 *PWM waveforms with and without blanking time T_Δ.*

The effects of blanking time on the output voltage is illustrated in *Figure 5-6*. Instead of a pure sine wave, the blanking time introduces low order harmonic distortion (such as the 3^{rd}, 5^{th}, 7^{th}) at the zero crossing point of the current. Here, the load current is shown as a lagging sine wave.

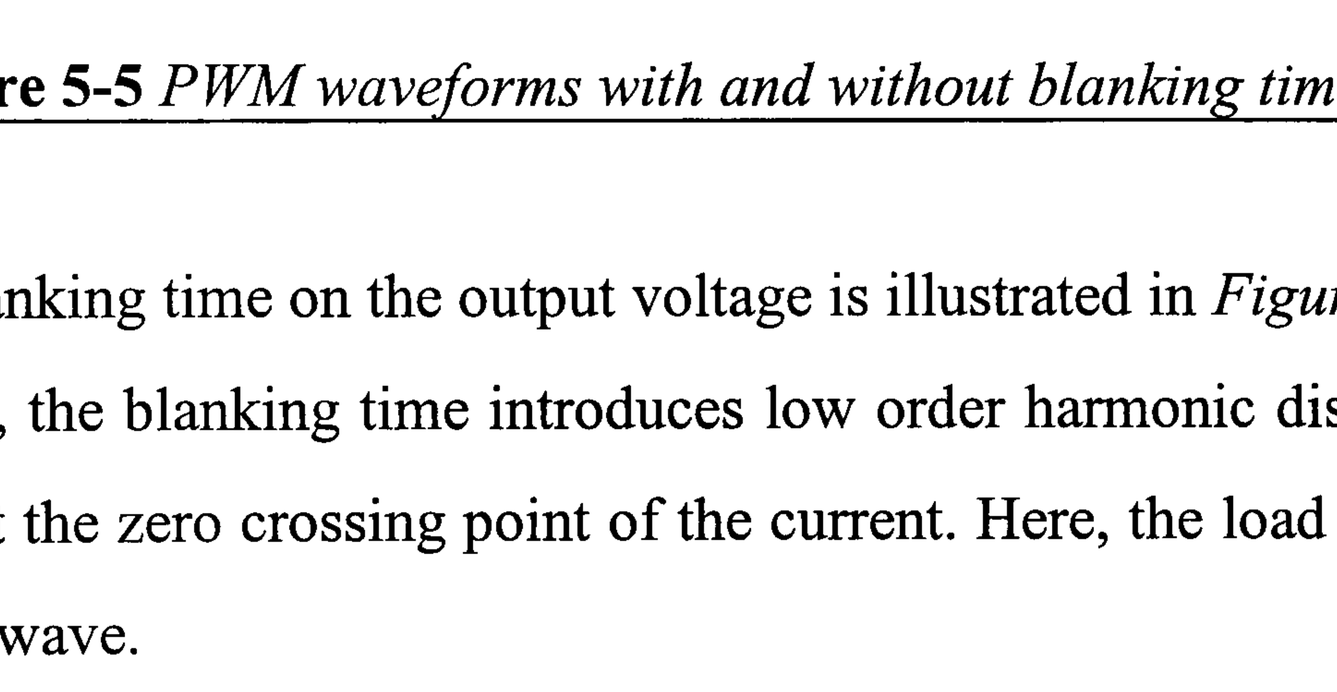


Figure 5-6 *The effects of blanking time, T_Δ, on the output voltage.*

In the TCAD simulation, blanking is achieved using the circuit shown in *Figure 5-7.* A delay unit and an AND gate are used to introduce blanking time into the PWM waveform. The parameter in the delay unit sets the length of the blanking time.

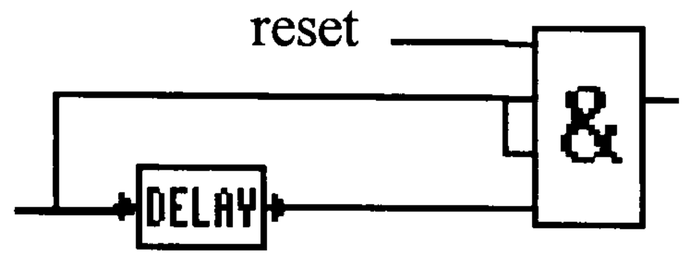


Figure 5-7 *Simulation of blanking time in TCAD.*

5.2.2 Protection Circuit

It is recognised that a short circuit can sometimes occur at the output terminals of the inverter with rather serious consequences. In the simulation, a simple protection circuit model, shown in *Figure 5-8*, is designed to analyse the condition. If a short circuit occurs at the terminals as modelled by the wire AA' in *Figure 5-9*, the voltage across the terminals becomes negligible. Therefore, the d.c. voltage Vs will fall across the two switching devices, resulting in a very high on-state V_{CE} (collector-emitter voltage) value. Voltage sensors were used to monitor the V_{CE} of devices during on-states. These voltages were compared against a threshold value in the comparator units and if they exceed the threshold value, the thyristor in the latching circuit (*Figure 5-8*) is latched on. This causes the output of the latching circuit, ERROR (active LOW), to be LOW, therefore disabling the signals to the switching devices. The inverter then ceases to operate. V_{CE} monitoring is a simple and cheap way of circuit protection because it does not require the use of current sensors. However, it does not protect the circuit against all fault conditions.

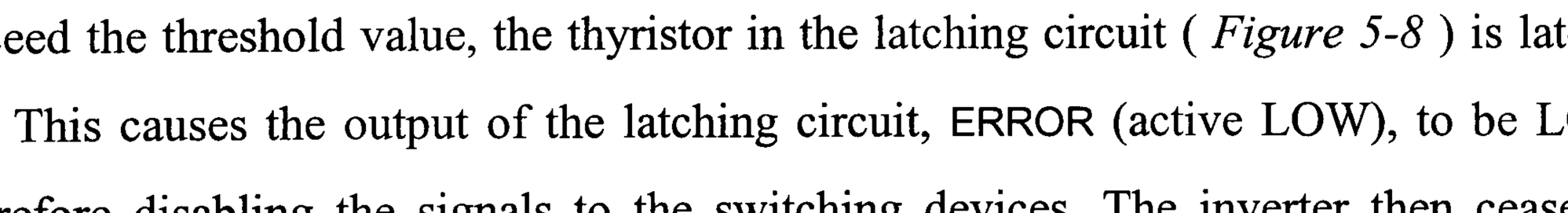


Figure 5-8 *Latching circuit for short-circuit protection.*

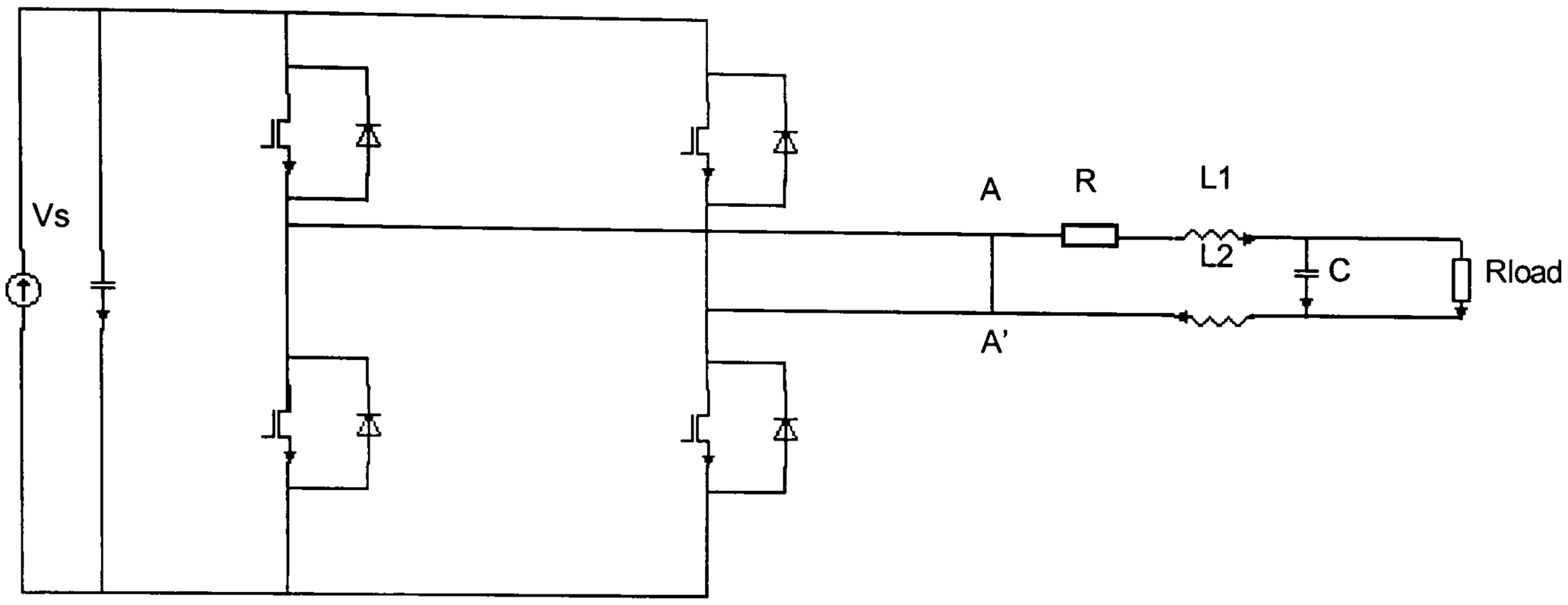


Figure 5-9 *Short circuit at AA'.*

5.2.3 Output Filter

In its original state, the PWM waveform contains a lot of high order harmonic distortions. To obtain a sinusoidal voltage from the distorted waveform, a low pass L-C filter is required to eliminate the high order harmonics. The transfer function of the filter is given by

Eq. 5-3
$$\frac{v_o}{v_i} = \frac{1}{1 + j\varpi L\left(\frac{1}{Z_L} + j\varpi C\right)}$$

Eq. 5-4
$$\frac{v_o}{v_i} = \frac{1}{1 - \varpi^2 LC + j\frac{\varpi L}{Z_L}}$$

where Z_L is the load impedance.

By assuming that $Z_L \to \infty$,

Eq. 5-5
$$\frac{v_o}{v_i} = \frac{1}{1 - \varpi^2 LC}$$

The cut-off frequency of the filter is given by

Eq. 5-6 $$\omega^2 LC = 1$$

$$(2\pi f_o)^2 LC = 1$$

Eq. 5-7 $$f_o = \frac{1}{2\pi\sqrt{LC}}$$

Harmonic components with frequencies above the cut-off frequency are attenuated by a second order fall-off in gain. However, a closer examination of the transfer function reveals that the LC filter is not truly a low pass filter with a cut-off frequency at f_O . From *Eq. 5-5*, it can be seen that the magnitude of the filter gain at f_O is infinity. *Figure 5-10* shows the gain of the filter. Frequency components at and around the cut-off frequency (which is also the resonant frequency) are grossly amplified. In practice, the circuit impedance will prevent an infinite gain, but frequencies around f_O will still be amplified to a considerable degree.

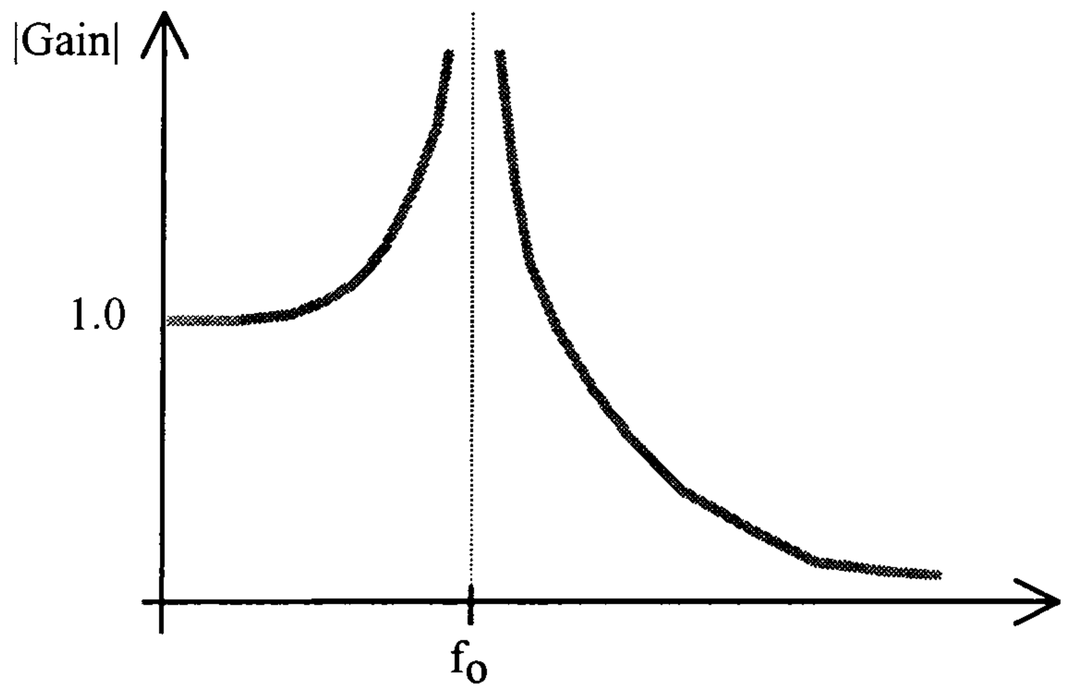


Figure 5-10 *Gain of the LC filter.*

The values chosen for L and C are L = 2 mH and C = 33 μF.

From *Eq. 5-7* these values can be shown to give a cut-off frequency at f_O = 619Hz, which is roughly the 12th harmonic (assuming the fundamental frequency is 50Hz). Harmonic distortions of a pure PWM waveform appear around the *n*th harmonic where $n = m_f$, (m_f+1), (m_f+2), etc. For m_f= 400 (i.e. f_s = 50 Hz, f_{tri} = 20kHz), distortions will start around the 400th harmonic. Therefore, this LC combination will eliminate all the high order harmonics. The complete model of the inverter and PWM controller is shown

in *Figure 5-11*. It incorporates the blanking time modules, the protection circuit and the output filter.

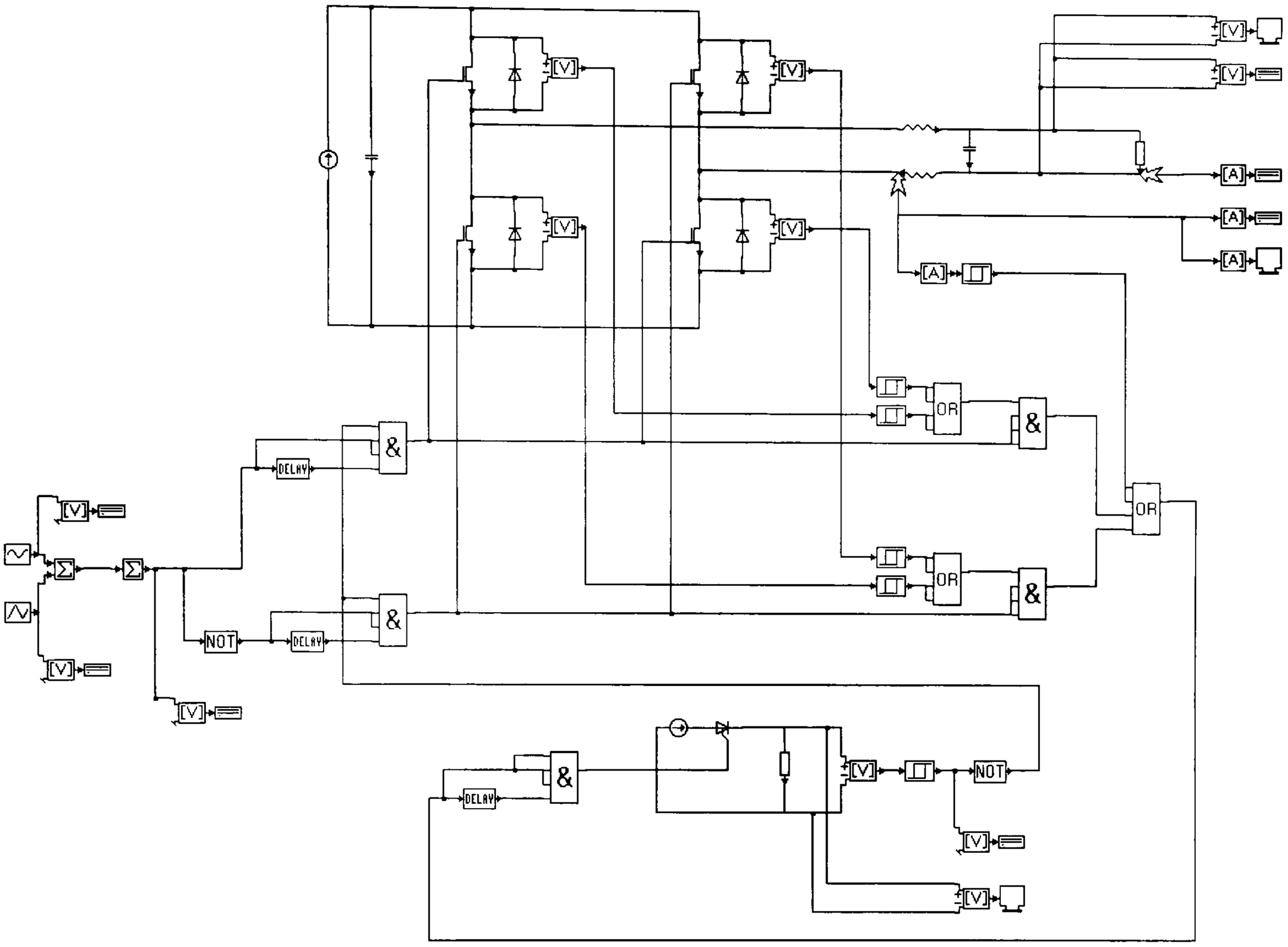


Figure 5-11 *TCAD model of a PWM inverter.*

5.3 Simulation results

Two sets of PWM waveforms are required for the operation of the single phase inverter: one set to control T1 & T3 and the other set to control T2 & T4. The inverter is designed with bipolar voltage switching [63] therefore the second PWM waveform is just the inverse of the first waveform (neglecting blanking time).

5.3.1 Without Blanking

Figure 5-12 shows the filtered output voltage of the PWM Inverter before blanking was introduced. The waveform contains some transient distortions particularly in the first cycle. A TCADHar spectrum analysis of the waveform (see *Figure 5-13*) shows that the distortions are caused by elements of the 12^{th} harmonic and its sidebands. This is

because the LC filter has a resonant frequency near the 12th harmonic of the output voltage. However, in steady state the voltage settles into a smooth 50Hz sinusoidal wave, with negligible harmonics.

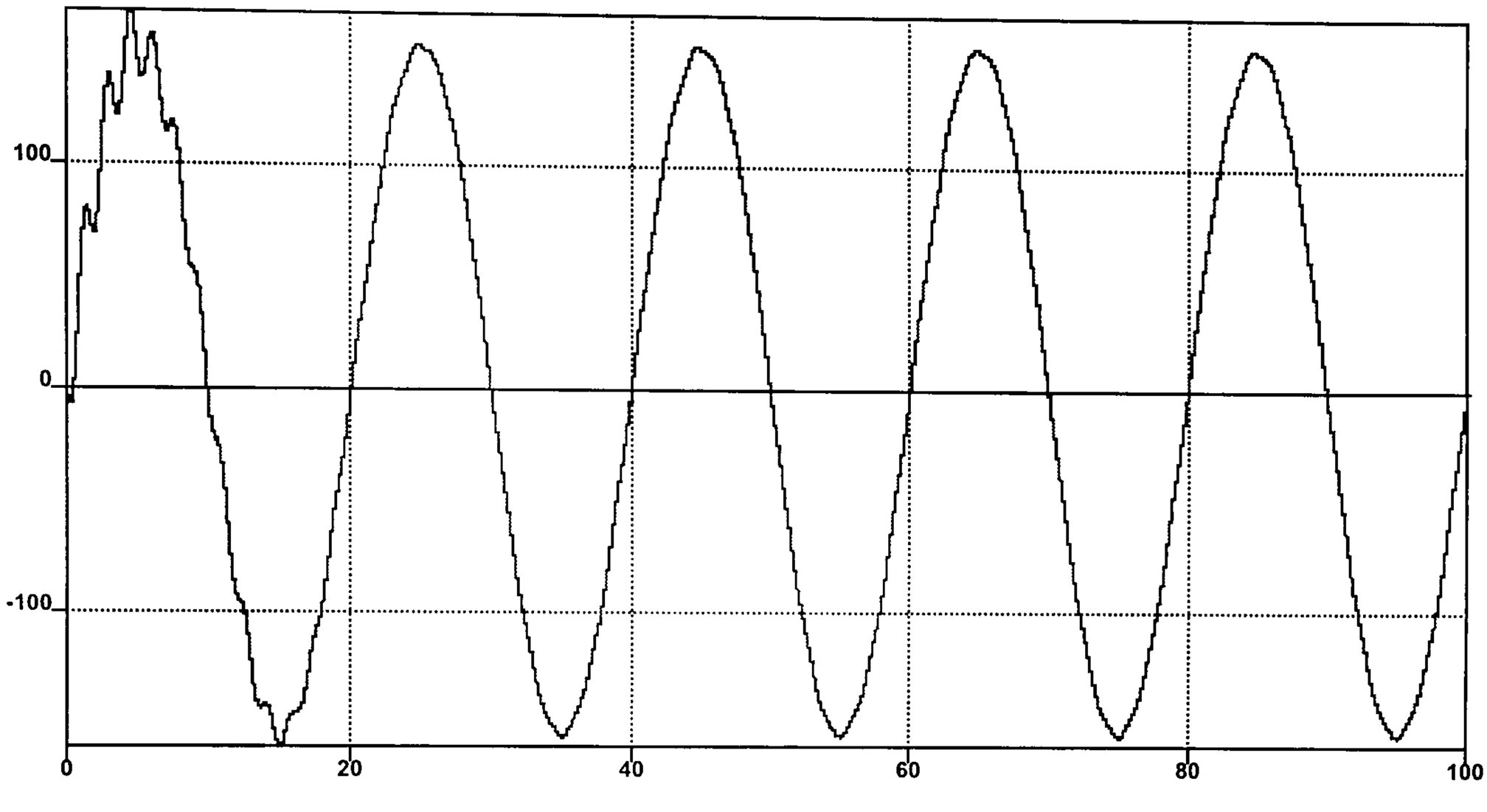


Figure 5-12 *PWM Inverter output voltage after filtering (without blanking time).*

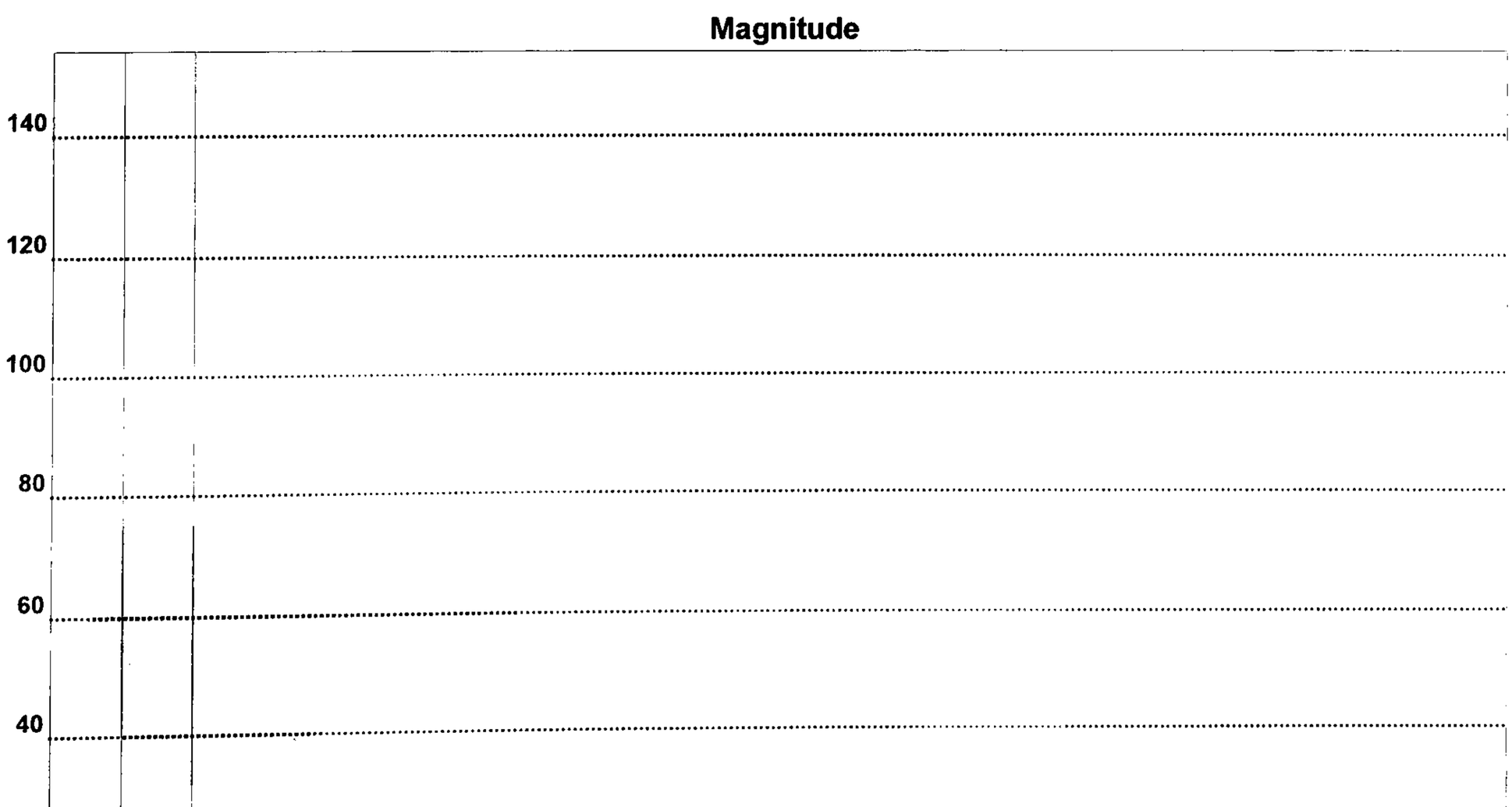


Figure 5-13 *Spectrum analysis of the filtered output voltage (without blanking time).*

5.3.2 With 2.7μs of Blanking Time

A PWM Inverter with 2.7μs of blanking time was simulated in TCAD and the results. *Figure 5-14* shows the filtered output voltage of the inverter. The value for the blanking time was chosen based on the practical limitations and features of the devices used in the design. Blanking introduces 3rd harmonics into the waveform. However, transient effects of the LC filter are reduced as the spectrum analysis shows negligible 12th harmonics but about 4.6% of 3rd harmonic distortion (see *Figure 5-15*). This harmonic component can be automatically eliminated in line-to-line voltage of three phase inverters.

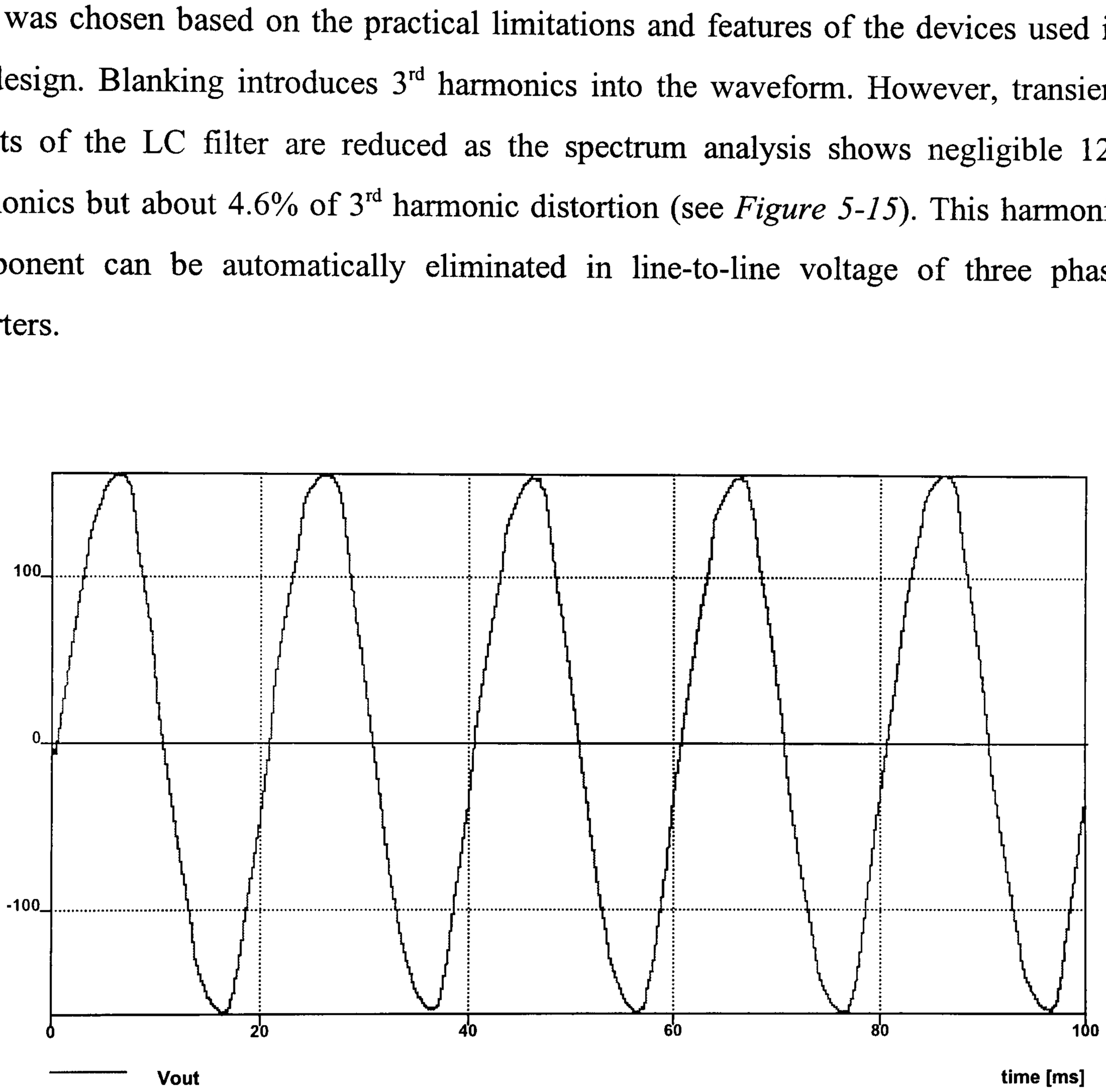


Figure 5-14 *PWM Inverter output voltage after filtering (with 2.7μs blanking time) .*

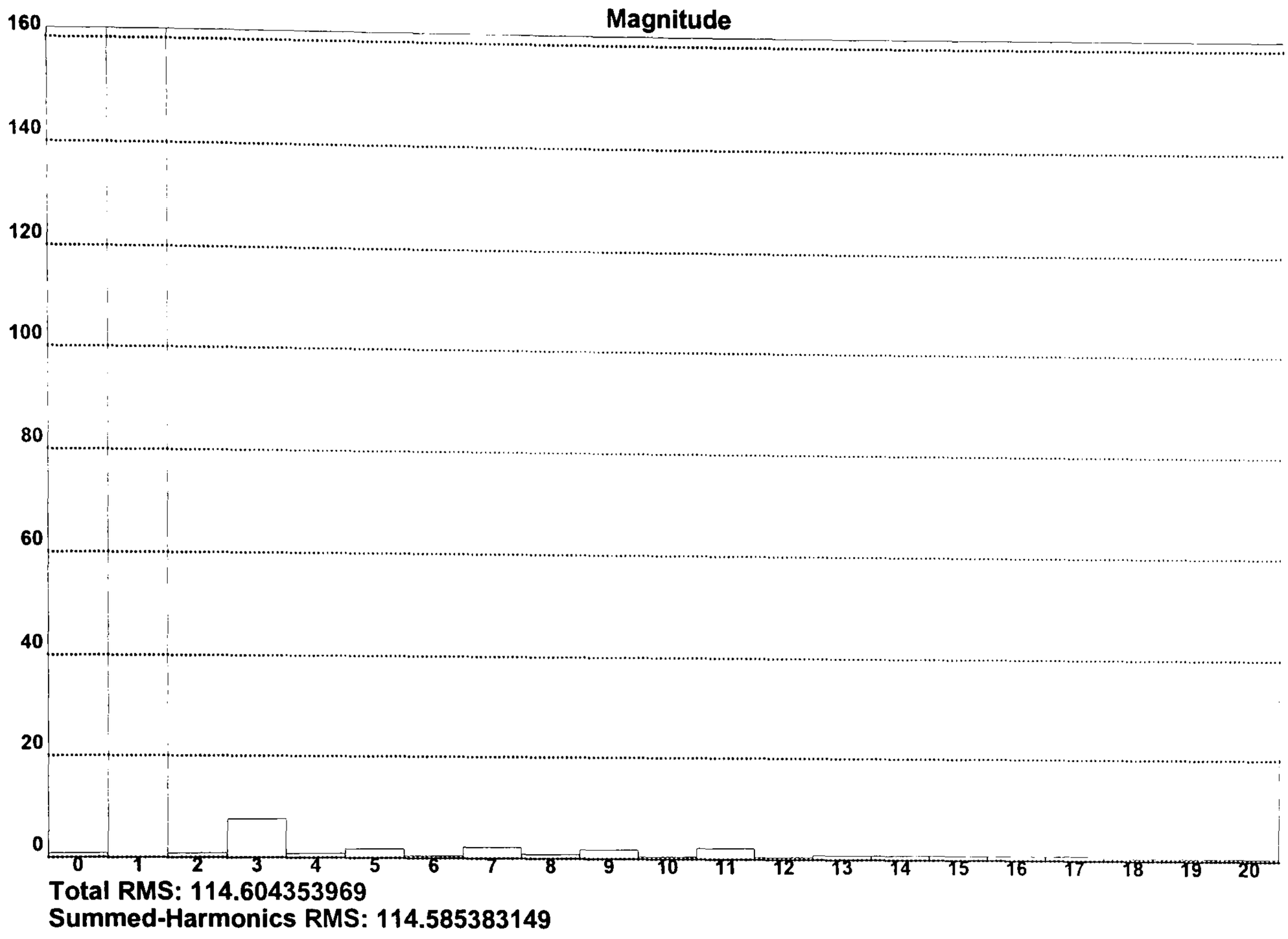


Figure 5-15 *Spectrum analysis of the filtered output voltage (2.7μs blanking time).*

5.3.3 Protection Circuit

A protection circuit for the inverter was designed and simulated as described earlier. If a short circuit occurs at the inverter terminals as shown in *Figure 5-9*, the total d.c. voltage Vs will be across the two devices, hence V_{CE} = Vs/2. This will trigger the V_{CE} monitoring circuit, therefore cutting off all switching pulses to the devices and unless the voltage across the capacitor in the LC filter at that instant is zero, oscillation will occur in the LC configuration. A resistor R=1.0Ω was added into the circuit to provide a small resistive load to prevent the simulation from producing an undamped oscillation. In practice, the wiring and filter will have a finite resistance value.

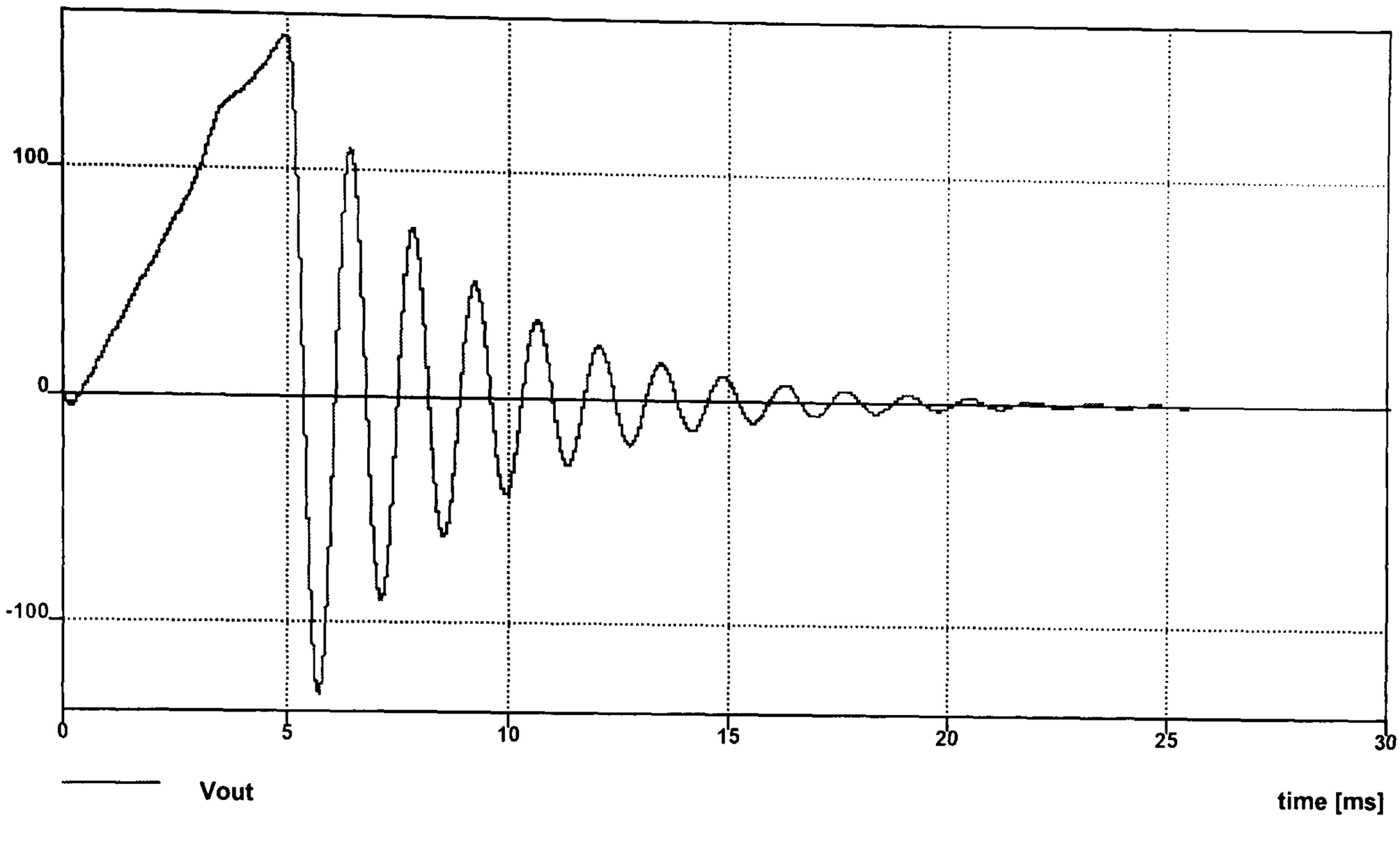


Figure 5-16 *Short circuit fault at 5ms.*

5.4 Three Phase Inverter

The circuit of a three phase PWM inverter, as illustrated by *Figure 5-17* , was designed and simulated in TCad. The three phase PWM waveform was generated by three separate units, each comparing a triangular waveform with a sine wave. The sine waves of the three units are phase shifted by 120°.

Simulation results for the three phase inverter show that the circuit suffers from a longer transient distortion than the single phase inverter, as shown in *Figure 5-18.* It takes about 350ms for the line-to-line voltage to settle into a sine wave. This simulation was performed using the same filter values as that used for the single phase inverter. All the simulations described are based on the original sine wave PWM method. This method can be implemented using relatively simple and cheap hardware design. After checking the performance, it was decided to implement the design in hardware.

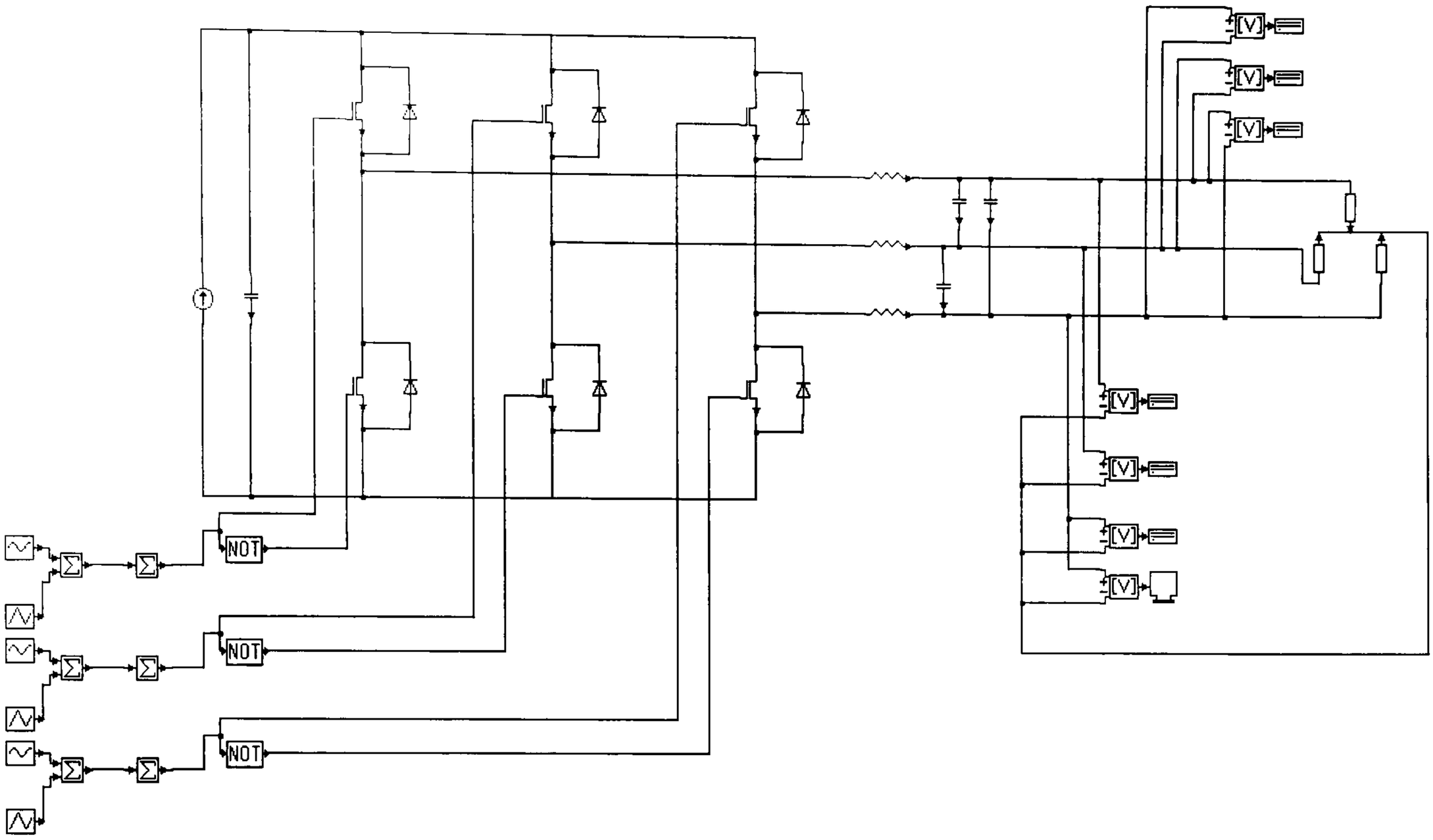


Figure 5-17 *TCad schematic of a three phase inverter and control circuitry.*

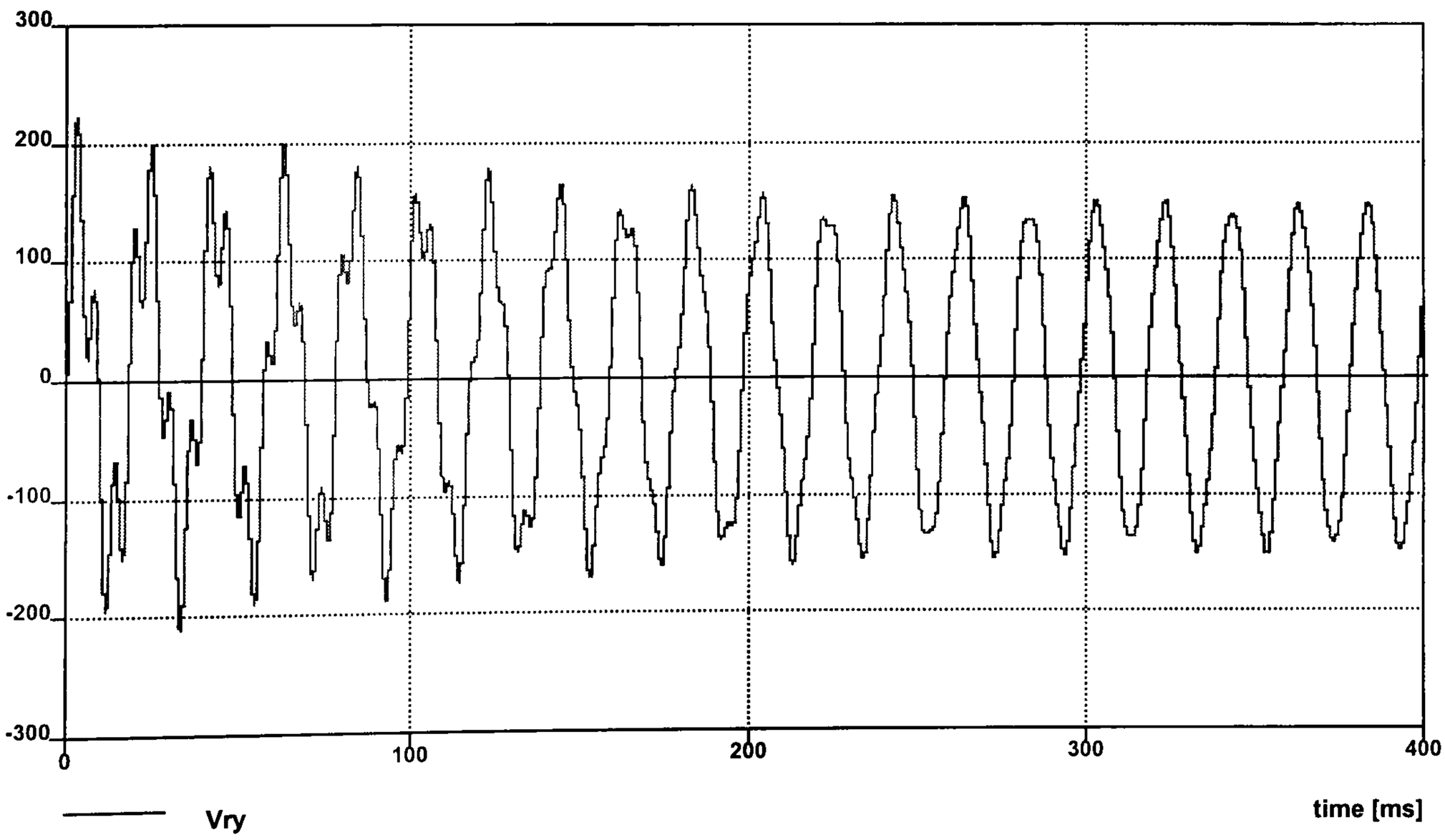


Figure 5-18 *Line-to-line voltage of a three phase inverter.*

5.5 Hardware Implementation

This section describes the circuit implementation for two designs of PWM inverter control. In the first design, a C-program is used to generate the switching pattern which is subsequently stored in an EPROM. The second design uses a dedicated integrated circuit (IC) to generate the pattern.

5.5.1 EPROM based PWM controller

There are various design solutions to implement a PWM controller. A fairly straight forward method is to use an Erasable Programmable Read Only Memory (EPROM) to store the PWM switching patterns. During the operation, this information is sequentially retrieved and fed into a driver circuit board which will switch the IGBTs accordingly. *Figure 5-19* shows a schematic of the circuit design. It comprises a voltage controlled oscillator NE566 (IC1), a counter (IC2), an EPROM (IC3) and some AND gates (IC4) to act as output buffers.

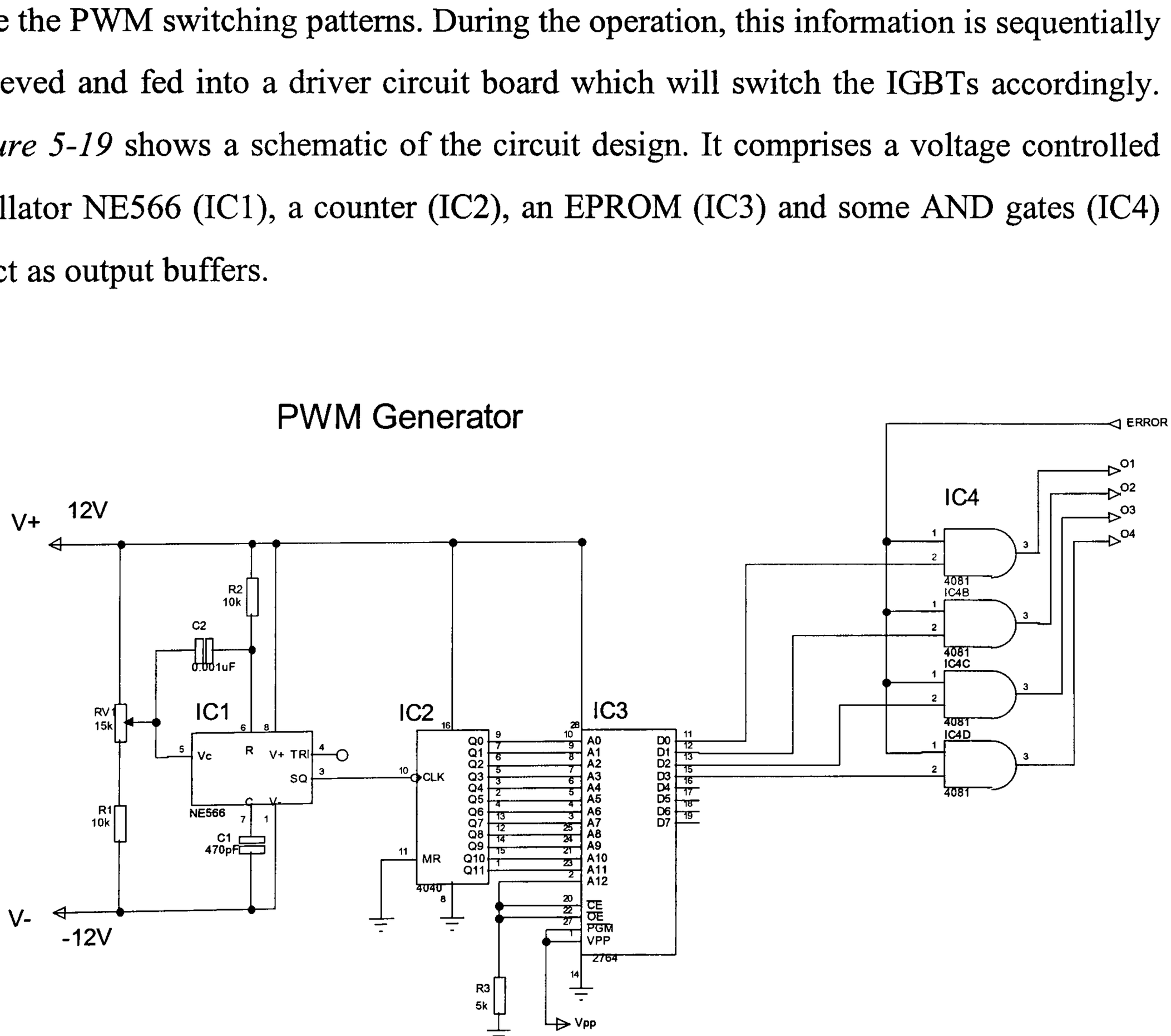


Figure 5-19 *Circuit diagram of the EPROM based PWM generator.*

The switching pattern in the EPROM is generated using a C++ program. The information for producing one cycle of the power waveform, i.e. one period of the sinusoidal reference signal, is broken down into 4096 slices and stored in the EPROM memory locations. Each memory location corresponds to an address ranging from 0 to 4095 and each bit of information in a memory location controls one power switch in the inverter. For a single phase inverter which has 4 power switches, 4096x4 bits (16kb) of memory are required while a three phase inverter with 6 switches requires 4096x6 bits (24kb) of memory.

IC2 is a CMOS4040 12-bit counter, designed to count from 0 to 4095 in a repetitive cycle. This is used as the address input to retrieve information from the EPROM. To obtain an output frequency of 50Hz, the counter (IC2) has to complete 50 cycles in one second. Therefore, the sampling frequency has to be :

Eq. 5-8 $f_s = 50 \times 4096 = 204.8$ kHz

The advantage of using a voltage controlled oscillator instead of a fixed frequency oscillator is that a voltage signal can be used to control the oscillator frequency and hence the sampling frequency of the inverter. This makes it possible to control the inverter frequency with a closed loop control circuit. Due to immediate availability during the implementation stage, a 64kb-EPROM is used in the circuit although 16kb ($2^{12} \times 4$) of information is sufficient for single phase operation (24kb for three phase). The period of the triangular carrier wave is chosen to contain 10 sampling units. Each sampling unit corresponds to one clock cycle hence the actual sampling time will be the inverse of the clock frequency.

In the C program, a comparison between the reference power waveform and the carrier waveform is made at every sampling point. The output is 1 if the reference power value is larger than the carrier value and 0 if vice versa. The necessary switching signal is generated from this comparison as shown in *Figure 5-20.* However, as a result of introducing discrete sampling points, a certain amount of error is inevitable. The errors are labelled as $\pm \varepsilon_n$ in the diagram. The maximum value for each error is just under the length of one sampling unit which, in this case, is 10% of the period of the switching

signal (because one cycle of the switching signal consists of 10 sampling units). The effects of these errors can be reduced by increasing the number of sampling points in each cycle of the switching signal.

This can be done either by

1. maintaining the frequency modulation ratio m_f and increasing the total number of sampling points in the power cycle

or

2. maintaining the number of sampling points in one power cycle and reducing m_f..

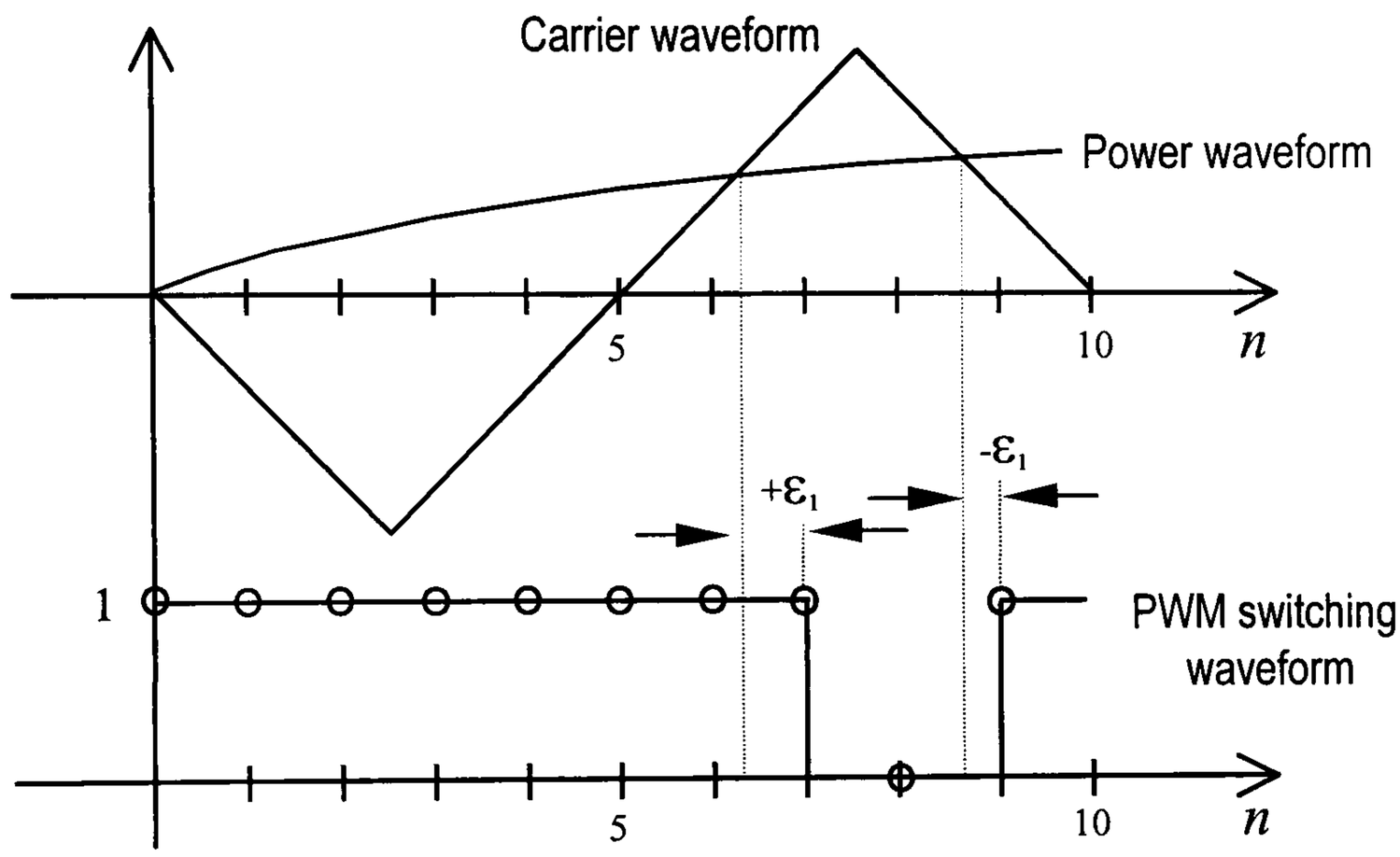


Figure 5-20 *Generation of PWM waveform.*

The frequency of the triangular carrier waveform, also known as the switching frequency is given by:

Eq. 5-9 $$f_{tri} = 1/(NT_s)\ \text{Hz}$$

where

T_s is the sampling period which is determined by the desired power frequency and N is the number of sampling points in one cycle of the carrier signal.

From *Eq. 5-8* , for a 50Hz power frequency, the sampling frequency T_s is:

$$T_s = \frac{1}{50} \cdot \frac{1}{4096}$$

$$T_s = 4.88\mu s$$

Therefore, from *Eq. 5-9*, the switching frequency is

$$f_{tri} = \frac{1}{10 \times 4.88 \times 10^{-6}} \text{Hz}$$

$$f_{tri} = 20.48 \text{ kHz}$$

The frequency modulation factor is given by,

$$m_f = \frac{f_{tri}}{f_1} = \frac{(1/4096T_s)}{(1/10T_s)} = 409.6$$

where

f_{tri} is the frequency of the triangular carrier waveform (switching frequency)

f_1 is the frequency of the fundamental harmonic (sinusoidal power frequency)

A three phase PWM waveform generator was also constructed by simply changing the contents of the EPROM with a new C program which is written to generate the three phase switching data. In the program, the triangular carrier waveform is compared with three different sinusoidal power waveform, each phase shifted from one another by 120°. The result of each comparison determines the switching signal of the IGBTs in each branch of the inverter. Instead of four outputs, the three phase PWM controller has 6 outputs, as there are 6 IGBTs in a three phase inverter. Therefore, two additional data outputs from the EPROM are used.

5.5.2 SA828 PWM Generator

The SA828 is a three phase PWM waveform generator from *GEC Plessey*. It is designed for applications such as uninterruptible power supplies, variable speed control of a.c. machines and a range of other inverter operations. The SA828 has six TTL level PWM outputs which control the six switches in a three phase inverter via a driver circuit.

Functional Description

Figure 5-21 shows a simplified internal block diagram of the SA828 [64]. The PWM fundamental waveform (reference waveform) is stored in an on-chip ROM. It can be a standard sinusoidal waveform, a sine + third harmonic waveform or any symmetrical wave shape made to order. This project uses the device with a standard sinusoidal reference waveform which is sometimes referred to as the *power waveform* in the data sheet [64].

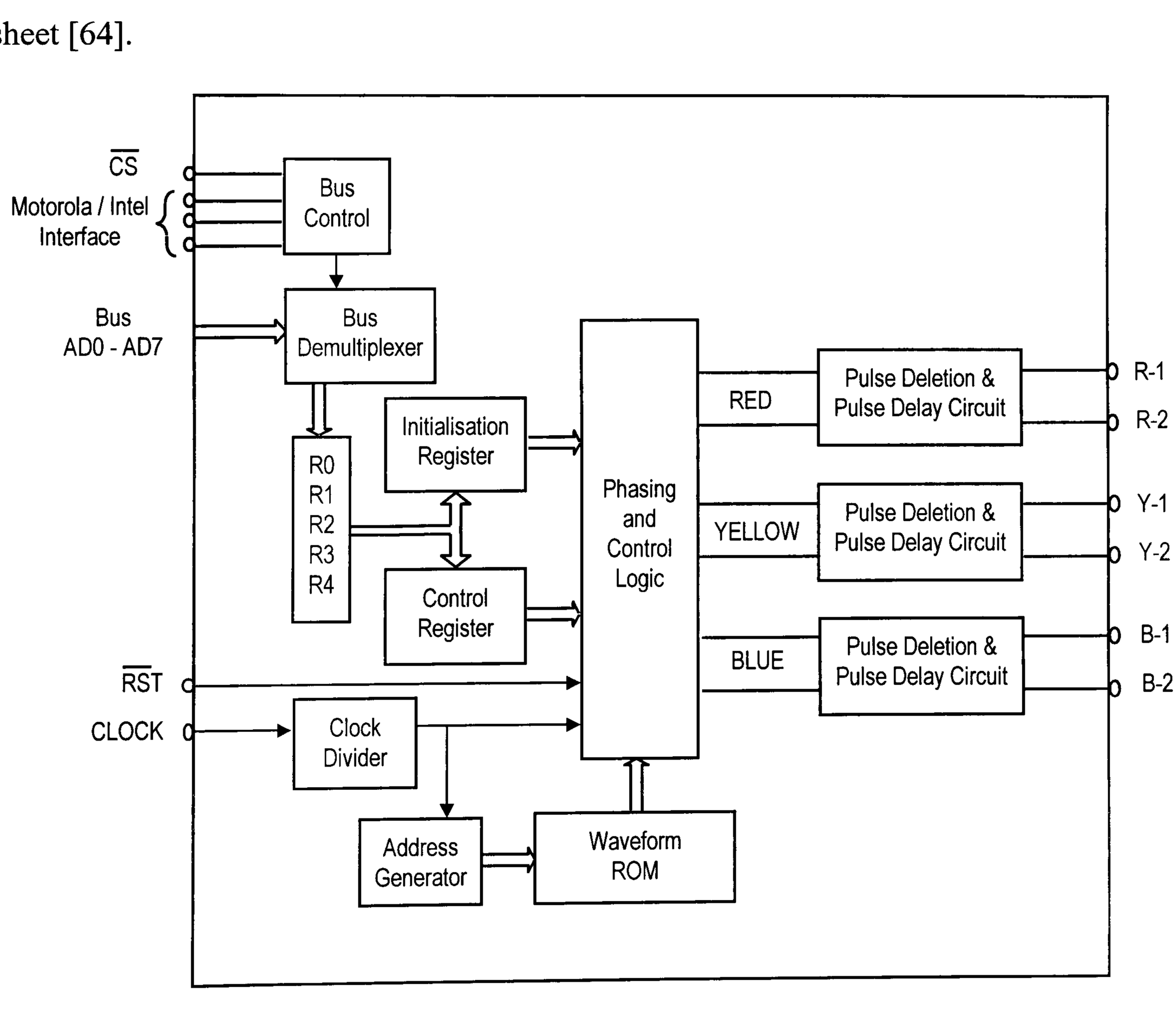


Figure 5-21 *Simplified internal block diagram of SA828.*

Unlike the EPROM solution previously described where the content of the memory is a set of switching patterns generated by a computer program, this device stores only the shape of the power waveform and not the actual switching pattern. Calculation of the switching pattern is performed on-chip in the *Phasing and Control Logic* block. The advantage of this approach is the ability to change PWM parameters such as the modulation ratios during the generation of the switching pattern. In the EPROM solution, this would require going back to the programming stage, which eliminates any chance of real-time manipulation.

Using the maximum clock frequency of 12.5MHz, the triangular carrier frequency can be selected up to 24kHz. The SA828 is designed to operate in conjunction with an external microprocessor. Values of two groups of registers have to be programmed into the device in order to set the parameters for the desired output waveform. The communication protocol between the SA828 and the microprocessor can either be Intel or Motorola bus timing.

Controlling the SA828

The SA828 is controlled by loading data into two 24-bit registers, the *Initialisation register* and the *Control register*, via a microprocessor interface. The initialisation register contains all the information on the parameters which are constant. The parameters set in the initialisation register are:

1. Carrier frequency
2. Power frequency range
3. Pulse delay time
4. Pulse deletion time
5. Counter reset

The data in the initialisation register is loaded only once, during power up, and does not change in the course of the operation. Data in the control register, however, can be modified during the operation of the chip, allowing real-time control of the PWM waveform.

The parameters set in the control register are

1. Power frequency
2. Overmodulation
3. Forward / Reverse
4. Output inhibit
5. Power waveform amplitude

The data loaded into the chip's System Bus is first stored in the temporary registers R0 to R2. Then, if the register R4 is loaded, the data is transferred to the Initialisation Register. Otherwise, if register R3 is loaded, the data is transferred to the Control Register. *Figure 5-22* shows a flow chart of the routine.

Calculations

The choices for the PWM parameters are based on the desired output waveform as well as the overall system requirements. The following calculations are used to determine the necessary data for the initialisation and control registers. They are based on the information provided in the data sheet [64]. The clock frequency used in this application is 12MHz.

Initialisation Register

The initialisation register controls five PWM parameters. They are:

1. The Carrier Frequency f_{carr} is set at 11.7kHz.

This is the frequency of the triangular waveform which is compared to the reference waveform to obtain the PWM switching pattern.

$$f_{carr} = \frac{f_{clk}}{512 \times n}$$

$$\Rightarrow n = \frac{f_{clk}}{512 \times f_{carr}} = \frac{12 \times 10^6}{512 \times 11.7 \times 10^3} = 2$$

where f_{carr} = carrier frequency and f_{clk} = clock frequency

The carrier frequency is determined by a 3-bit word, denoted by CFS, during initialisation. From the data sheet, when n=2, CFS = 001.

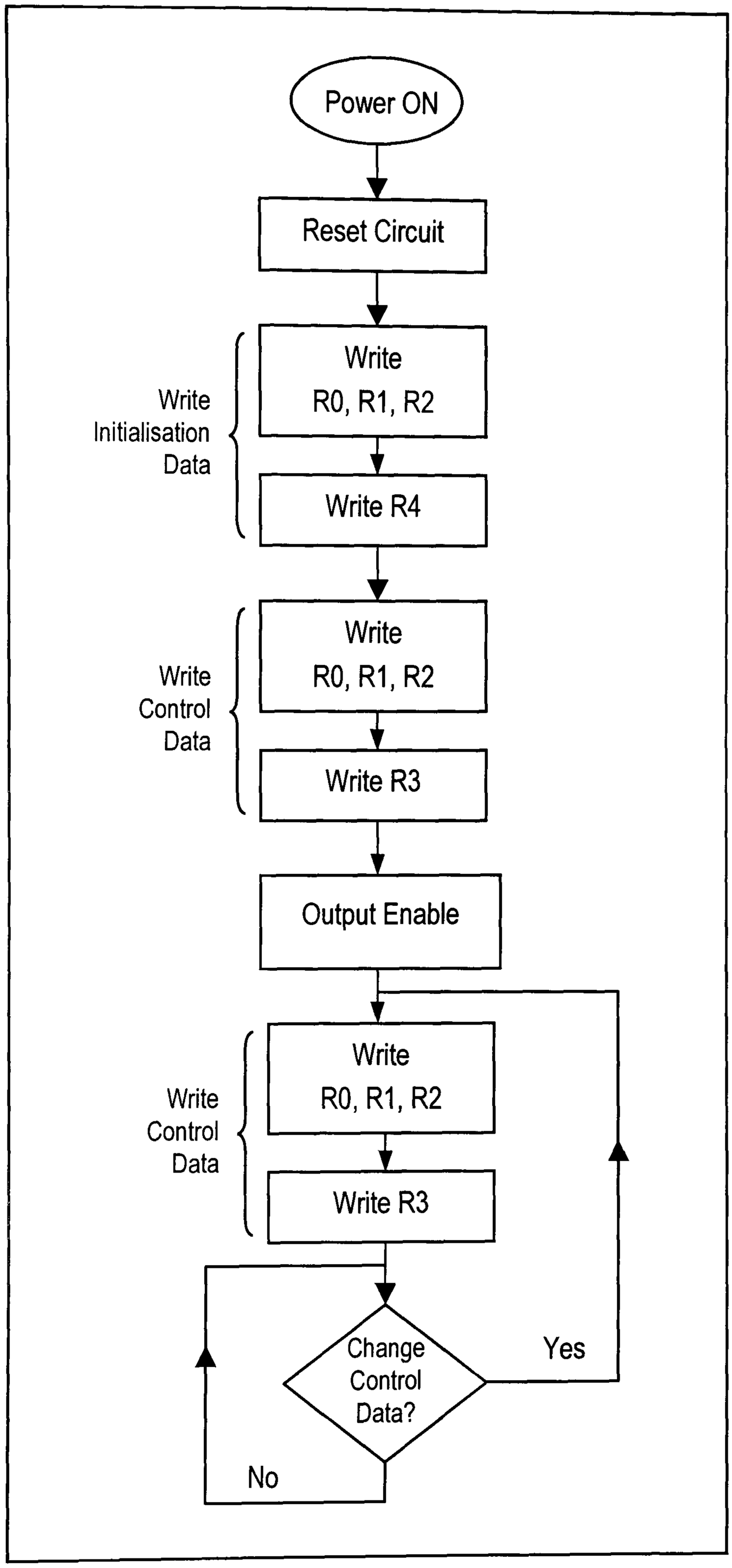


Figure 5-22 *Flowchart of the SA828 data loading routine.*

2. The Reference Frequency Range f_{range} is set for 61 Hz .

This parameter sets the maximum reference frequency of the waveform. In variable speed motor applications, this feature prevents the machine from being operated outside its design parameters.

$$f_{range} = \frac{f_{carr} \times m}{384}$$

$$\Rightarrow m = \frac{f_{range} \times 384}{f_{carr}} = \frac{61 \times 384}{11.7 \times 10^3} = 2$$

The reference frequency range is determined by the 3-bit word denoted by FRS. From the data sheet, when m=2, FRS = 001.

3. The Pulse Delay Time t_{delay} is set at 5.83μs.

The pulse delay time is the blanking time between two complementary switching waveforms. Setting a large delay time increases low harmonic distortion to the output waveform. At the same time, the delay has to be large enough to prevent a 'shoot through' between two complementary power switches.

The pulse delay time is determined by a 6-bit word, PDY.

$$t_{delay} = \frac{pdy}{f_{carr} \times 512}$$

$$\Rightarrow pdy = t_{delay} \times f_{carr} \times 512 = 5.83 \times 10^{-6} \times 11.7 \times 10^3 \times 512 = 35$$

where pdy is the value of the PDY word.

4. The Pulse Deletion Time t_{pd} is set at 10μs.

Theoretically, the pulses in a PWM waveform can be infinitesimally narrow. However, in practice, a narrow pulse may cause problems to the power switches due to storage effects. The pulse deletion time sets a minimum pulse width time. Pulses with widths narrower than the minimum time are eliminated altogether.

From the data sheet [64]:

$$t_{pd} = \frac{pdt}{f_{carr} \times 512}$$

$$\Rightarrow pdt = t_{pdt} \times f_{carr} \times 512 = 10 \times 10^{-6} \times 11.7 \times 10^{3} \times 512 = 60$$

PDT is the 7-bit word which determines the pulse deletion time and pdt is the value of PDT. From the data sheet, when pdt = 60, PDT =1001110.

5. Counter reset (active low).

This facility allows the internal reference frequency counter to be set to zero. When the Counter Reset is active LOW, the internal reference frequency phase counter is set to 0° for the red phase. The Counter Reset is released when HIGH. For this application the counter is set HIGH all the time.

Control Register

The control register provides on-line control over four parameters. They are:

1. Reference Frequency (Power Frequency)

The reference frequency is the frequency of the inverter's output voltage. The present application does not require variable frequency output. The reference frequency is set at a constant value of 50Hz by the 8-bit word PFS in the Control Register. The decimal value of the word is denoted by pfs.

$$f_{power} = \frac{f_{range} \times pfs}{4096}$$

$$pfs = \frac{f_{power} \times 4096}{f_{range}} = \frac{50 \times 4096}{61} = 3357$$

From the data sheet, when pfs = 3357, PFS = 00011011.

2. Overmodulation

Overmodulation is not required, therefore the overmodulation switch OM is set to 0.

3. Reference Waveform Amplitude (Power Waveform Amplitude)

The reference waveform amplitude is set in percentage of the maximum value. For this application, the reference waveform amplitude is required to be maintained constant even if disturbances cause the d.c. line voltage to change. This means that on-line control of the reference waveform amplitude setting is required to maintain a constant output. However, for initial testing purposes, the amplitude setting of this open loop control circuitry is set at 80%. Hence the amplitude in percentage is,

$$A_{power} = \frac{A}{255} \times 100\%$$

$$\Rightarrow A = \frac{A_{power} \times 255}{100} = \frac{80 \times 255}{100} = 204$$

where A is the decimal value of the 7-bit word AMP which determines the Reference Waveform Amplitude. From the data sheet, when A = 204, AMP =1001100

4. Inhibit

Output inhibit is set to '0' during the first control sequence and set to '1' during subsequent control sequence.

From the preceding set of calculations, a table containing the register data can be obtained as shown in *Table 5-2*. The first column shows the register while the third column in the far left shows the hexadecimal value of the register.

It was mentioned that the SA828 is designed to be used in conjunction with a microprocessor. There is also a possibility of designing an FPGA-based digital circuit using VHDL to operate together with the SA828. This option is further discussed in *Chapter 9*. The present design employ the use of a PIC16x00 Series microcontroller to control the SA828. The microcontroller is programmed to feed the necessary signals into the PWM IC in the correct format. A listing of the program code is included in *Appendix C*.

Table 5-2 *Values for Initialisation and Control Registers.*

Initialisation

	CR	PDT_6	PDT_5	PDT_4	PDT_3	PDT_2	PDT_1	PDT_0	
R0	1	1	0	0	1	1	1	0	CE
	FRS_2	FRS_1	FRS_0	X	X	CFS_2	CFS_1	CFS_0	
R1	0	0	1	1	1	0	0	1	39
	X	X	PDY_5	PDY_4	PDY_3	PDY_2	PDY_1	PDY_0	
R2	1	1	1	0	0	0	0	1	E1

Control 1

	PFS_7	PFS_6	PFS_5	PFS_4	PFS_3	PFS_2	PFS_1	PFS_0	
R0	0	0	0	1	1	0	1	1	1B
	F/R	OM	INH	X	PFS_{11}	PFS_{10}	PFS_9	PFS_8	
R1	0	0	0	1	1	1	0	1	1D
	AMP_7	AMP_6	AMP_5	AMP_4	AMP_3	AMP_2	AMP_1	AMP_0	
R2	1	1	0	0	1	1	0	0	CC

Control 2

	PFS_7	PFS_6	PFS_5	PFS_4	PFS_3	PFS_2	PFS_1	PFS_0	
R0	0	0	0	1	1	0	1	1	1B
	F/R	OM	INH	X	PFS_{11}	PFS_{10}	PFS_9	PFS_8	
R1	0	0	1	1	1	1	0	1	3D
	AMP_7	AMP_6	AMP_5	AMP_4	AMP_3	AMP_2	AMP_1	AMP_0	
R2	1	1	0	0	1	1	0	0	CC

In this chapter, various issues regarding PWM inverters and their control systems were discussed. In addition, the design of two electronic control circuits for PWM control were presented. These control circuits were successfully implemented and used in the experiments in *Chapter 8*. The measured waveforms can be found in *Appendix E*. The next chapter looks at the development of another substantial component of the project's control system, the *fuzzy logic controller*.

Chapter
6

Fuzzy Logic Controller

Over the past few years, the use of fuzzy set theory, or fuzzy logic, in control systems has been gaining widespread popularity, especially in Japan. From as early as the mid-1970s, Japanese scientists have been instrumental in transforming the theory of fuzzy logic into a technological realisation. Today, fuzzy logic based control systems, or simply, *Fuzzy Logic Controllers* (FLCs) can be found in a growing number of products, from washing machines to speedboats, from air condition units to hand-held autofocus cameras. In the present work, fuzzy logic is employed in the speed governing system of the synchronous generator set.

Sections 6.1 and *6.2* of this chapter present an introduction to the theory of fuzzy logic control systems. All the terms and definitions introduced here are used extensively in the remaining sections of the chapter to describe the work on the FLC designed for the generator system in study.

6.1 Introduction to Fuzzy Logic

The success of fuzzy logic controllers is mainly due to their ability to cope with knowledge represented in a linguistic form instead of representation in the conventional mathematical framework. Control engineers have traditionally relied on mathematical models for their designs. However, the more complex a system, the less effective the mathematical model. This is the fundamental concept that provided the motivation for

fuzzy logic and is stated by Lofti Zadeh, the founder of fuzzy set theory, as the *Principle of Incompatibility*.

Zadeh stated that [65]:

"As the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics..."

Real-world problems can be extremely complex and complex systems are inherently fuzzy. Precision increases fuzziness. The main advantage of fuzzy logic controllers is their ability to incorporate experience, intuition and heuristics into the system instead of relying on mathematical models. This makes them more effective in applications where existing models are ill-defined and not reliable enough.

6.1.1 Historical Review

The term 'fuzzy' in fuzzy logic was first coined in 1965 by Professor Lofti Zadeh, then Chair of UC Berkeley's Electrical Engineering Department. He used the term to describe multivalued sets in the seminal paper, 'Fuzzy Sets' [66]. The work in his paper is derived from multivalued logic, a concept which emerged in the 1920s to deal with Heisenberg's *Uncertainty Principle* in quantum mechanics. Multivalued logic was further developed by distinguished logicians such as Jan Lukasiewicz, Bertrand Russell and Max Black. At the time, multivalence was usually described by the term 'vagueness'. When Zadeh developed his theory, he introduced the term 'fuzzy'.

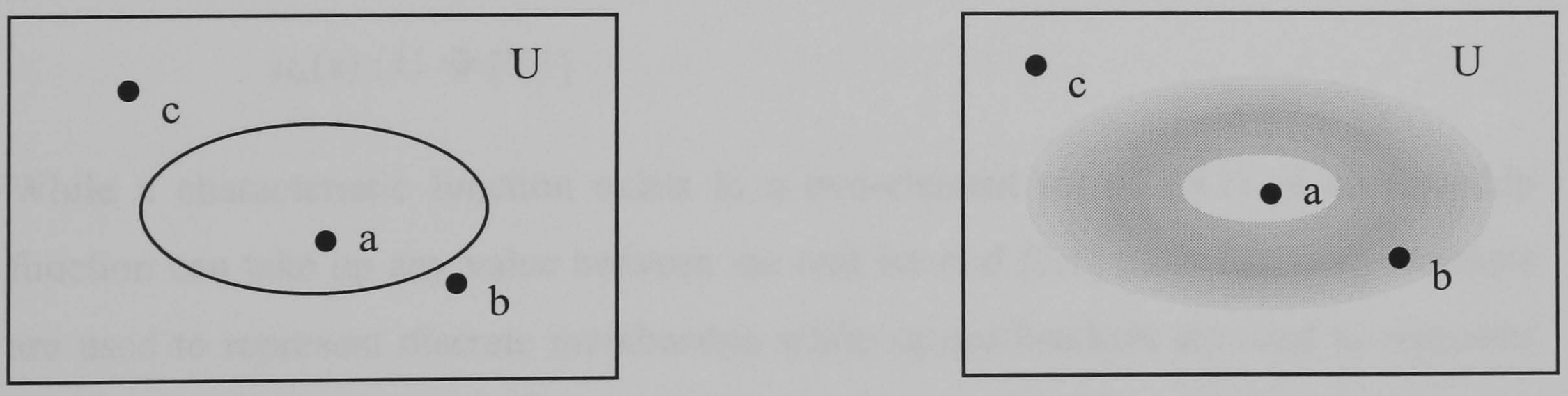
Zadeh applied Lukasiewicz's multivalued logic to set theory and created what he called *fuzzy sets* - sets whose elements belong to it in different degrees. According to the *fuzzy principle*, 'everything is a matter of degree'. While conventional logic is bivalence (TRUE or FALSE, 1 or 0), fuzzy logic is multivalence (*from* 0 to 1). It is a shift from conventional mathematics and number crunching to philosophy and language. At the beginning, fuzzy logic remained very much a theoretical concept with little practical applications. The work Zadeh was involved in consisted mostly of computer simulations of mathematical ideas. In the 1970s, Professor Ebrahim Mamdani of Queen's Mary

College, London, built the first fuzzy system, a steam engine controller, and later the first fuzzy traffic lights. This led to the extensive development in fuzzy control applications and products which is evident today.

6.1.2 Fuzzy Sets & Fuzzy Logic

Classical Sets & Fuzzy Sets

Classical set theory was founded by the German mathematician, Georg Cantor (1845-1918). In the theory, a *universe of discourse* U, is defined as a collection of objects all having the same characteristics. A classical set is then a collection of a number of those elements. The member elements of a classical set belong to the set one hundred percent. Other elements in the universe of discourse, which are non-member elements of the set, are not related to the set at all. A definitive boundary can be drawn for the set, as depicted in *Figure 6-1(a)*.



(a) (b)

Figure 6-1 *(a) Classical/Crisp set boundary (b) Fuzzy set boundary.*

A classical set can be denoted by

$$A=\{x \in U \mid P(x)\}$$

where the elements of A has the property P , and U is the universe of discourse.

Definition: *Characteristic function.*

$$\mu_A(x) : U \rightarrow \{0,1\}$$

The characteristic function $\mu_A(x)$ is defined as '0' if x is not an element of A and '1' if x is an element of A. Here, U contains only two elements, '1' and '0'. Therefore, an element x, in the universe of discourse is either a member of set A or **not** a member of set A. There is no ambiguity about membership. For example, consider the set ADULT, which contains elements classified by the variable AGE. It can be said that an element with AGE='5' would not be a member of the set whereas an element with AGE='45' would be. The question which arises is, where can a sharp and discrete line be drawn in order to separate members from non-members? At AGE='18'? By doing so, it means that elements with AGE='17.9' are not members of the set ADULT but those with AGE='18.1' are. This system is obviously not realistic to model the definition of an adult human. Simple problems such as this one embody the notion behind Zadeh's Principle of Incompatibility.

In fuzzy set theory, the concept of characteristic function is extended into a more generalised form, known as *membership function.*

Definition: *Membership Function.*

$$\mu_A(x) : U \rightarrow [0,1]$$

While a characteristic function exists in a two-element set of {0,1}, a membership function can take up any value between the unit interval [0,1] (note that curly brackets are used to represent discrete membership while square brackets are used to represent continuous membership). The set which is defined by this extended membership function is called a *fuzzy set.* In contrast, a classical set which is defined by the two-element characteristic function, as described earlier, is called a *crisp set.* Fuzzy set theory essentially extends the concept of sets to encompass *vagueness*. Membership to a set is no longer a matter of 'true' or 'false', '1' or '0', but a matter of degree. The degree of membership becomes important. The boundary of a fuzzy set is shown in *Figure 6-1(b).* While point *a* is a member of the fuzzy set and point *c* is not a member, the membership of point *b* is ambiguous as it falls on the boundary. The concept of *membership function* is used to define the extent to which a point on the boundary belongs to the set.

Definition: *Fuzzy Set.*

A fuzzy set F can be defined by the set of tuples

$$F=\{ (\mu_F(x),x) \mid x \in U \}$$

Zadeh proposed a notation for describing fuzzy sets whereby '+' denotes enumeration and '/' denotes a tuple.

Therefore, the fuzzy set F in Zadeh's notation becomes

$$F = \int_U \mu_F(x) / x \quad \text{for a continuous universe U}$$

or

$$F = \sum_{x \in U} \mu_F(x) / x \quad \text{for a discrete universe U.}$$

Returning to the earlier example, an element with AGE='18.1' may now be assigned with the membership degree to the set ADULT of, say, 1.0. An element of AGE='17.9' may then have a membership degree of 0.8 instead of 0. Such gradual change in the degree of membership provides a better representation of the real world. However, the exact shape of the membership function is very subjective and depends on the designer and the context of the application.

While set operations such as complement, union and intersection are straight forward definitions in classical set theory, their interpretation is more complicated in fuzzy set theory due to the graded attribute of membership functions. Zadeh [66] proposed the following definitions as an extension to the classical operations :

Definition: *Fuzzy Set Operations*

Complement $\quad \forall x \in X : \mu_{A'}(x) = 1-\mu_A(x)$

Union $\quad \forall x \in X : \mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(y)]$

Intersection $\quad \forall x \in X : \mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(y)]$

These definitions form the foundations of the basics of fuzzy logic theory and will contribute towards a better understanding of the application of fuzzy logic in the thesis.

Earlier in this chapter, the concept of membership function was introduced. The relation between an element in the universe of discourse and a fuzzy set is defined by its membership function. The exact nature of the relation will depend on the shape or the type of membership function used.

6.1.3 Types of Membership Functions

Figure 6-2 shows various types of membership functions which are commonly used in fuzzy set theory. The choice of shape depends on the individual application. In fuzzy control applications, Gaussian or bell-shaped functions and S-functions are not normally used. Functions such as Γ-function, L-function and Λ-function are far more common.

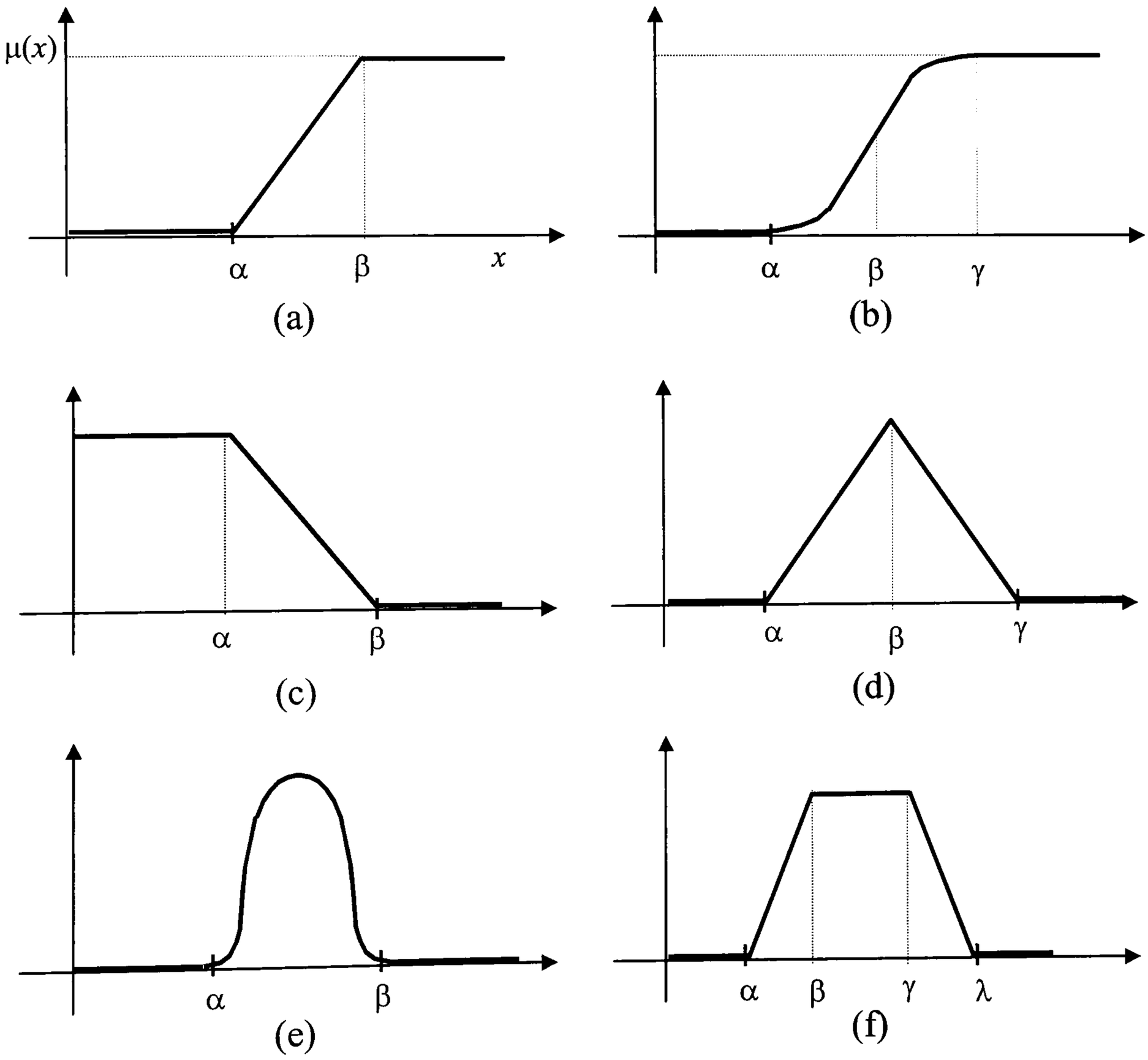


Figure 6-2 *Types of membership functions: (a)Γ-function (b)S-function (c)L-function (d)Λ-function (e)Gaussian function (f)Π-function.*

The definitions of the membership functions relevant to this thesis are as follows:

Γ-function, $\Gamma : U \rightarrow [0,1]$

$$\Gamma(x;\alpha,\beta) = \begin{cases} 0 & x < \alpha \\ (x - \alpha)/(\beta - \alpha) & \alpha \le x \le \beta \\ 1 & x > \beta \end{cases}$$

L-function, $L : U \rightarrow [0,1]$

$$L(x;\alpha,\beta) = \begin{cases} 1 & x < \alpha \\ (x - \beta)/(\alpha - \beta) & \alpha \le x \le \beta \\ 0 & x > \beta \end{cases}$$

Λ-function, $\Lambda : U \rightarrow [0,1]$

$$\Lambda(x;\alpha,\beta,\gamma) = \begin{cases} 0 & x < \alpha \\ (x - \alpha)/(\beta - \alpha) & \alpha \le x \le \beta \\ (x - \gamma)/(\beta - \gamma) & \beta \le x \le \gamma \\ 0 & x > \gamma \end{cases}$$

6.1.4 Linguistic Variables

The concept of a *linguistic variable*, a term which is later used to describe the inputs and outputs of the FLC, is the foundation of fuzzy logic control systems. A conventional variable is numerical and precise. It is not capable of supporting the vagueness in fuzzy set theory. By definition, a linguistic variable is made up of words, sentences or artificial language which are less precise than numbers. It provides the means of approximate characterisation of complex or ill-defined phenomena. For example, 'AGE' is a linguistic variable whose values may be the fuzzy sets 'YOUNG' and 'OLD'. A more common example in fuzzy control would be the linguistic variable 'ERROR', which may have *linguistic values* such as 'POSITIVE', 'ZERO' and 'NEGATIVE'.

In this thesis, the following conventions are used to define linguistic variables:

If X_i is a linguistic variable defined over the universe of discourse U where $x \in$ U then

LX_i^k	(for $k = 1,\ldots n$)	are the linguistic values X_i can take
n		is the number of linguistic values X_i have
$\mu_{LXi,k}(x)$		is the LX_i^k membership function for the value x
$\underline{LX_i}$		is the set containing LX_i^k , where $\underline{LX_i} = \{ LX_i^1, LX_i^2, \ldots Lx_i^n \}$.

In the recent example above,

X_1	is 'ERROR'
$n = 3$	is the number of linguistic values in X_1
LX_1^1	is 'POSITIVE'
LX_1^2	is 'ZERO'
LX_1^3	is 'NEGATIVE'

and, for $x = \{-1, 0, 1\}$:

$\mu_{LX1,1}(-1) = 0$; $\mu_{LX1,1}(0) = 0$; $\mu_{LX1,1}(1) = 1$

$\mu_{LX1,2}(-1) = 0$; $\mu_{LX1,2}(0) = 1$; $\mu_{LX1,2}(1) = 0$

$\mu_{LX1,3}(-1) = 1$; $\mu_{LX1,3}(0) = 0$; $\mu_{LX1,3}(1) = 0$

However, in some parts of the thesis, the symbol 'μ' is omitted when describing the membership function. Further explanation regarding this omission is included in *Section 6.3.4.*

6.1.5 Fuzzy Logic Operators

Logical connectives are also defined for fuzzy logic operations. They are closely related to Zadeh's definitions of fuzzy set operations. The following are four fuzzy operations which are significant to the thesis. R denotes the relation between the fuzzy sets A and B.

Negation

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

Disjunction

R: A OR B $\qquad \mu_R(x) = \max[\mu_A(x), \mu_B(x)]$

Conjunction

R: A AND B $\mu_R(x) = \min[\mu_A(x), \mu_B(x)]$

Implication

R: $(x = A) \rightarrow (y = B)$ IF x is A THEN y is B

Fuzzy implication is an important connective in fuzzy control systems because the control strategies are embodied by sets of IF-THEN rules. There are various different techniques in which fuzzy implication may be defined. These relations are mostly derived from multivalued logic theory. The following are some of the common techniques of fuzzy implication found in literature:

Zadeh's classical implication :

$\mu_R(x,y) = \max \{ \min[\mu_A(x),\mu_B(y)], 1-\mu_A(x) \}$

Mamdani's implication :

$\mu_R(x,y) = \min[\mu_A(x), \mu_B(y)]$

Note that Mamdani's implication is equivalent to Zadeh's classical implication when :

$\mu_A(x) \geq 0.5$ and $\mu_B(y) \geq 0.5$.

Godel's implication :

$$\mu_R(x,y) = \begin{cases} 1 & \mu_A(x) \leq \mu_B(y) \\ \mu_B(y) & \text{otherwise} \end{cases}$$

Lukasiewicz' implication :

$\mu_R(x,y) = \min \{1, [1-\mu_A(x)+\mu_B(y)]\}$

The differences in using the various implication techniques are described in [67]. It is fairly obvious by looking at the mathematical functions of the different implication techniques that Mamdani's technique is the most suitable for hardware implementaion. It is also the most popular technique in control applications and is the technique that is used in the present design.

6.2 Fuzzy Control Systems

Figure 6-3 shows the block diagram of a typical Fuzzy Logic Controller (FLC) and the system plant as described in [68]. There are five principal elements to a fuzzy logic controller:

Fuzzification module (Fuzzifier).

Knowledge base.

Rule base.

Inference Engine.

Defuzzification module (Defuzzifier).

Automatic changes in the design parameters of any of the five elements creates an adaptive fuzzy controller. Fuzzy control system with fixed parameters are non-adaptive. Other non-fuzzy elements which are also part of the control system include the sensors, the analogue-digital converters, the digital-analogue converters and the normalisation circuits. There are usually two types of normalisation circuits: one maps the physical values of the control inputs onto a normalised universe of discourse and the other maps the normalised value of the control output variables back onto its physical domain.

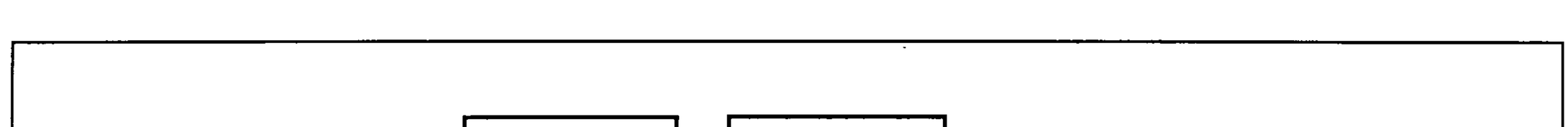


Figure 6-3 *Block diagram of a typical fuzzy logic controller.*

6.2.1 Fuzzifier

The fuzzification module converts the crisp values of the control inputs into fuzzy values, so that they are compatible with the fuzzy set representation in the rule base. The choice of fuzzification strategy is dependent on the inference engine, i.e. whether it is composition-based or individual-rule-firing based [69].

6.2.2 Knowledge base

The knowledge base consists of a data base of the plant. It provides all the necessary definitions for the fuzzification process such as membership functions, fuzzy set representation of the input-output variables and the mapping functions between the physical and fuzzy domain.

6.2.3 Rule base

The rule base is essentially the control strategy of the system. It is usually obtained from expert knowledge or heuristics and expressed as a set of IF-THEN rules. The rules are based on fuzzy inference concept and the antecedents and consequents are associated with linguistic variables. For example,

IF *error* (*e*) is *Positive Big* (PB) THEN *output* (*u*) is *Negative Big* (NB)

rule antecedent — rule consequent

Error (*e*) and *output* (*u*) are linguistic variables whilst *Positive Big* (PB) and *Negative Big* (NB) are the linguistic values. The rules are interpreted using a fuzzy implication technique. In fuzzy control theory, this is normally Mamdani's implication technique.

6.2.4 Defuzzifier

The diagram in *Figure 6-4* shows the membership functions related to a typical fuzzy controller's output variable defined over its universe of discourse. The FLC will process the input data and map the output to one or more of these linguistic values (LU^1 to LU^5). Depending on the conditions, the membership functions of the linguistic values may be clipped. *Figure 6-5* shows an output condition with two significant (clipped above zero) output linguistic values. The union of the membership functions forms the fuzzy output value of the controller.

This is represented by the shaded area in *Figure 6-5* and can be expressed by the fuzzy set equation:

$$S = \bigcup_{i=1}^{k} S_i$$

$$\mu_S(y) = \max_i[\mu_{S_i}(y)] \quad i = 1,2,\ldots k$$

where

S is the union of all the output linguistic values,

S_i is an output linguistic value with clipped membership function,

k is the total number of output linguistic values defined in the universe of discourse.

In most cases, the fuzzy output value S, has very little practical use as most applications require non-fuzzy (crisp) control actions. Therefore, it is necessary to produce a crisp value to represent the possibility distribution of the output. The mathematical procedure of converting fuzzy values into crisp values is known as 'defuzzification'.

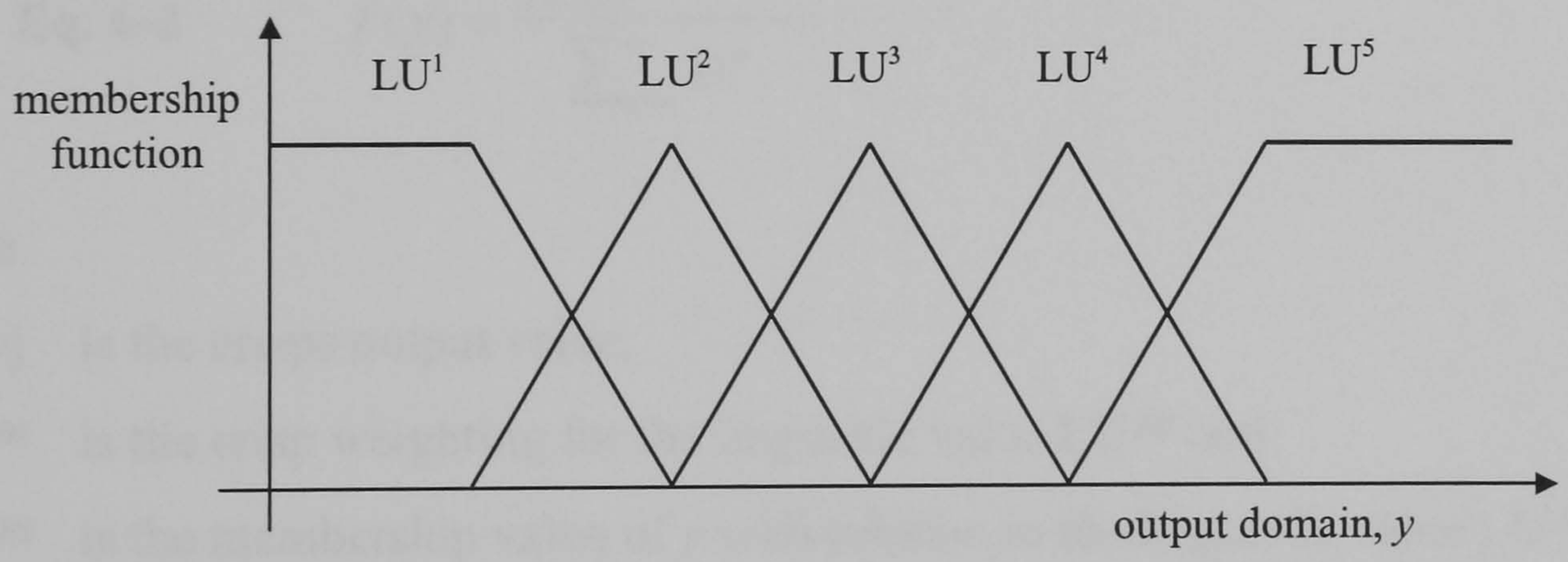


Figure 6-4 *Membership function of the output linguistic values.*

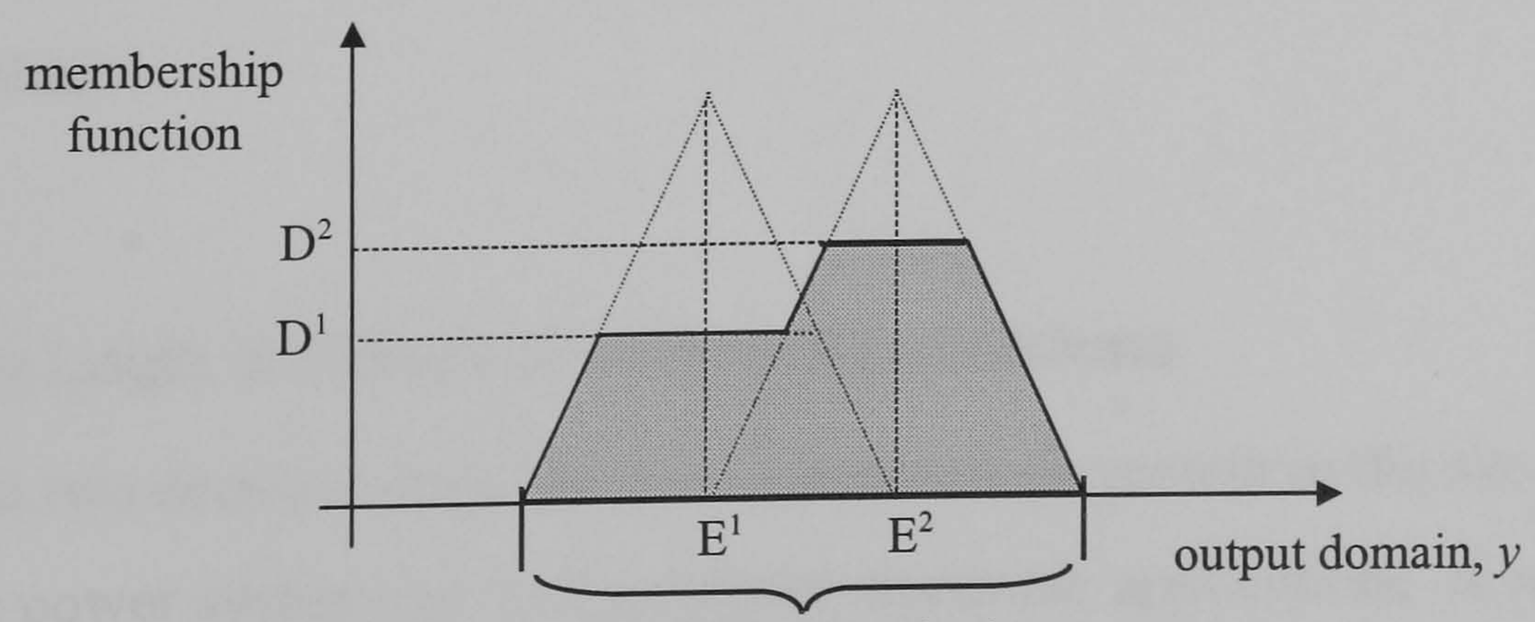


Figure 6-5 *Possibility distribution of an output condition.*

A number of defuzzification methods have been suggested. The different methods produces similar but not always the same results for a given input condition. The choice of defuzzification methods usually depends on the application and the available processing power.

The defuzzification method used in this thesis is the weighted average method. This method requires relatively little processing power and is ideal for FPGA implementation where 'area space' is a major consideration. However, it is only valid for symmetrical membership functions. Each membership function is assigned with a weighting, which is the output point where the membership value is maximum. Based on the diagram in *Figure 6-5*, the defuzzification process can be expressed by:

Eq. 6-1
$$f(y) = \frac{\sum \mu(y) \cdot y}{\sum \mu(y)}$$

and using the weighted average method on *Eq. 6-1* becomes:

Eq. 6-2
$$f(y) = \frac{\sum_{m=1}^{n} \mathrm{E}^m \cdot \mathrm{D}^m}{\sum_{m=1}^{n} \mathrm{D}^m}$$

where

$f(y)$ is the crisps output value,

E^m is the crisp weighting for the linguistic value LU^m and

D^m is the membership value of y with relation to the linguistic value LU^m.

The crisp output of the defuzzifier is used, either as it is, or via an interfacing block, to control the plant.

6.2.5 Fuzzy Logic in Power & Control Applications

Over the past two decades, there has been a tremendous growth in the use of fuzzy logic controller in power systems as well as power electronic applications. A recent series of tutorials in the IEE Power Engineering Journal [70,71,72] which focused entirely on the applications of fuzzy logic in power systems is evidence of its growing significance in the field. Current applications in power systems include power system stability control,

power system stability assessment, line fault detection and process optimisation for generation, transmission and distribution. In power electronics and control of electrical machines, fuzzy logic is used in motion control [73,74], control of wind turbines [75], motor efficiency optimisation control [76] and waveform estimation [77].

The advantages of using fuzzy logic in such applications include the following:

- Fuzzy logic controllers are not dependent on accurate mathematical models. This is particularly useful in power system applications where large systems are difficult to model. It is also relevant to smaller applications with significant non-linearities in the system.
- Fuzzy logic controllers are based on heuristics and therefore able to incorporate human intuition and experience.

There are numerous ways to build and implement a fuzzy logic system. It can either be based on a *fuzzy logic development shell* or built using software programming languages such as C++ or even Java. In this thesis, a different approach is made. The fuzzy system is developed using a hardware description language, VHDL.

6.3 Fuzzy Synchronous Generator Control

There are a number of reasons for using fuzzy logic in this application. The primary advantage is the wide range of flexibility fuzzy logic offers. As mentioned in *Section 6.2*, the backbone of any FLC is embodied in a set of fuzzy rules. The implication of this feature is two fold. The first point lies in the fact that the control strategy is represented by a set of rules and not an elaborated set of equations. This allows the designer to change the basic characteristics of the controller with minimal fuss, simply by redefining the rules. The structure of other components in the FLC remains intact and hence, the effort to redesign the hardware configuration is significantly reduced. This is a great advantage in research applications where some further studies into alternative control strategies are expected. Secondly, because the rules are *fuzzy*, vagueness in the design of the control system can be tolerated to a certain degree. This eliminates the need for a well-defined mathematical model of the plant. The control plant in this project is not a trivial system to model with significant accuracy, since it comprises a number of different non-linear components such as the diesel engine and synchronous generator. With reference to the equations developed in *Chapter 4,* models with reduced complexity can be employed for the purpose of analysing the functional characteristics of the FLC.

The following sections describe the work involved in developing the desired controller for the generator set based on the concepts introduced at the beginning of this chapter.

6.3.1 Control Block

Figure 6-6 shows a block diagram of the FLC and a stand-alone generator set. Components of the FLC are represented by the shaded blocks. There are two inputs to the FLC. Its final functionality is determined by the choice of inputs and the definition of the *fuzzy rule base*. This allows the system to be studied under different control strategies and specifications using the same hardware. Only three sections have to be modified: the two interfacing blocks and the rule base. In this experiment, the d.c. voltage at the output of the rectifier is used as an input to the FLC. The plan is to test the

system with a basic control strategy: maintaining the d.c. voltage within a small range, over varying load currents. The control output, u is used to control the rate of flow of the fuel to the diesel engine.

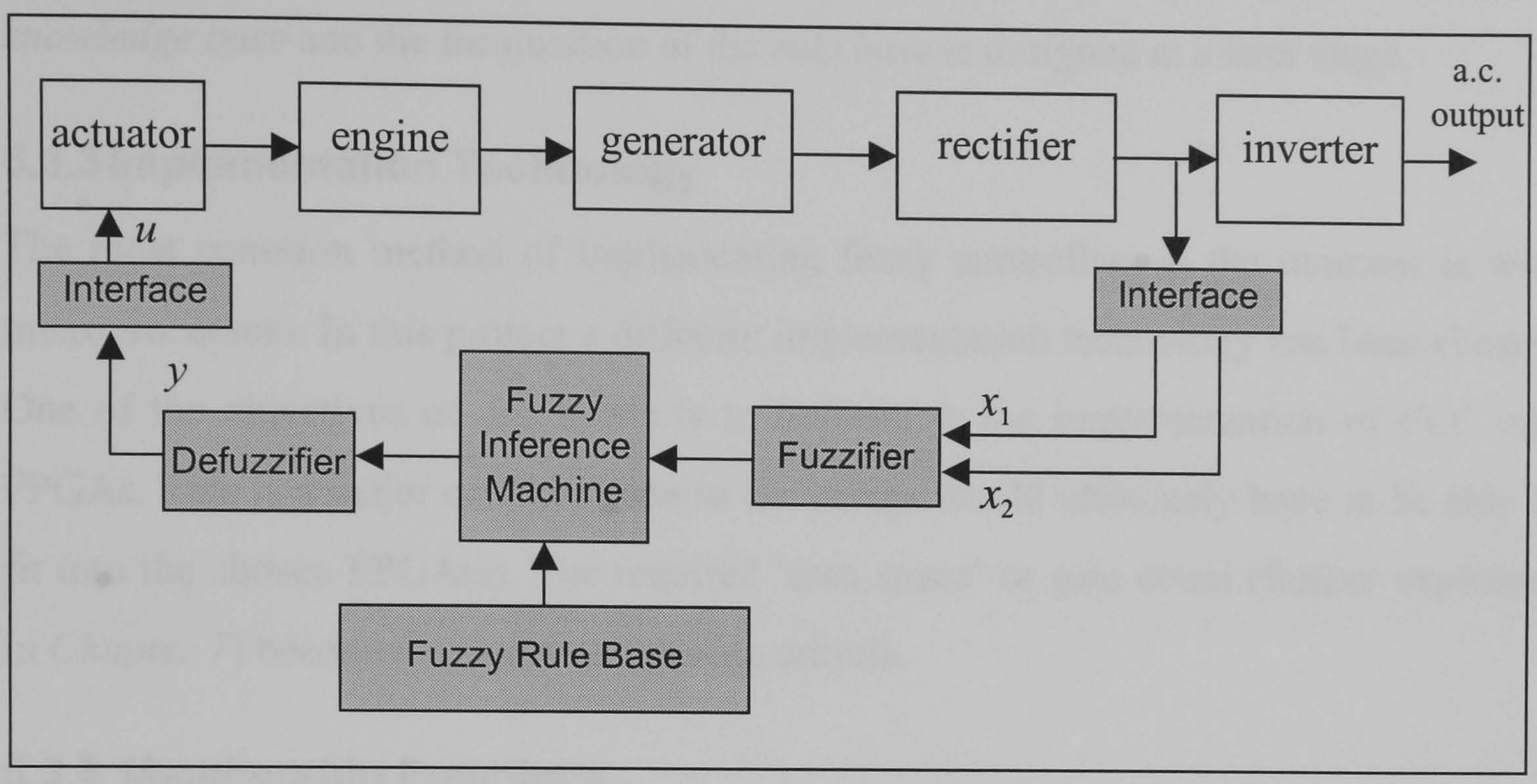


Figure 6-6 *Block diagram of Fuzzy Logic Controller and control plant.*

The tasks involved in designing a typical FLC can be loosely summarised as follows:

i. *Decide on an overall strategy based on the design criteria.*

ii. *Identify an implementation technology.*

iii. *Identify the I/O variables (knowledge base).*

iv. *Define the membership functions for the variables (knowledge base).*

v. *Formulate a Fuzzy Rule Base.*

vi. *Choose a method of inference.*

vii. *Choose a defuzzification technique.*

This design procedure forms a general guide and may well vary from one design to another, depending on the individual aims and requirements.

6.3.2 Overall Strategy

The design has to be able to accommodate different control strategies and yet not be too inherently complex for hardware implementation. Two input variables (x_1 and x_2) are used. This is enough to give the system a certain amount of possible capabilities. Too many inputs add complications to the design as the number of fuzzy rules expands

exponentially with the number of input variables. There are methods to handle large amount of input variables but these are not discussed in this thesis. A generic structure of the FLC is first designed and the specific functionality which is determined by the *knowledge base* and the formulation of the *rule base* is designed at a later stage.

6.3.3 Implementation Technology

The most common method of implementing fuzzy controllers at the moment is with microprocessors. In this project a different implementation technology has been chosen. One of the objectives of this thesis is to investigate the implementation of FLC into FPGAs. This is a major consideration as the design would ultimately have to be able to fit into the chosen FPGA(s). The required 'area space' or gate count (further explained in *Chapter 7*) becomes a significant design criteria.

6.3.4 Membership Functions

The present design utilises three types of functions : Γ-function, L-function and Λ-function, all of which have already been introduced. These functions have been proven to produce good results for control applications and can be easily implemented into hardware. The universe of discourse of the input variables is partitioned into 5 fuzzy sets or *linguistic values* (B^1 to B^5), whilst the output variable can take any of the 7 linguistic values (D_1 to D_7). *Figure 6-7* and *Figure 6-8* show graphical representations of the membership functions.

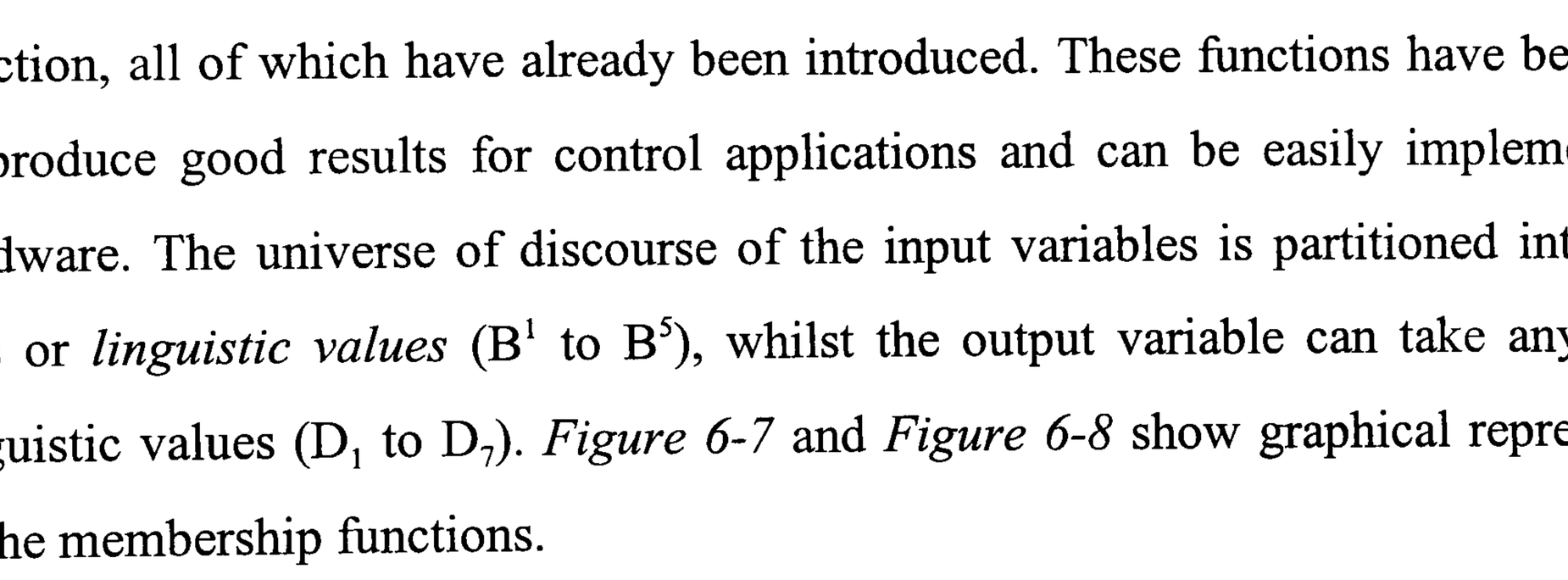


Figure 6-7 *Membership function of input variables.*

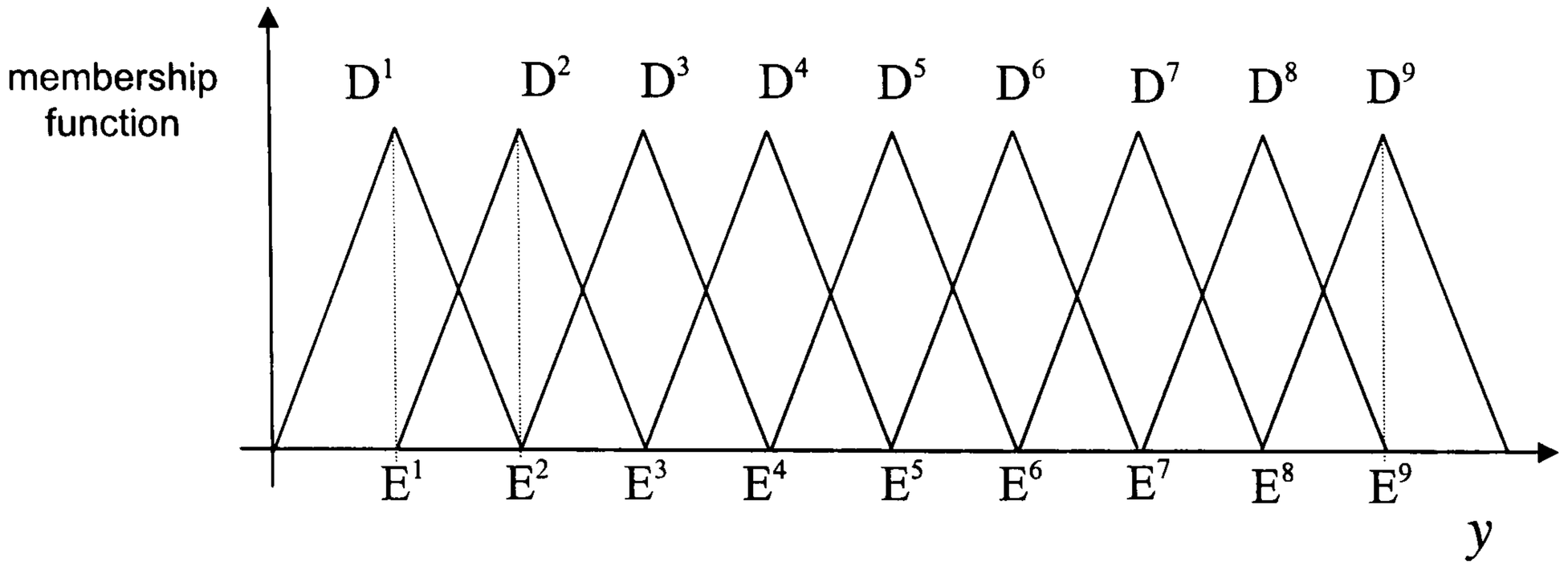


Figure 6-8 *Membership function of output variable.*

The following equations define the membership functions for the linguistic values associated with the input variables.

Eq. 6-3
$$\mathrm{B}_i^j = \begin{cases} 1 & x_i < a_i^j \\ \dfrac{(x_i - b_i^j)}{(a_i^j - b_i^j)} & a_i^j \le x_i \le b_i^j \\ 0 & x_i > b_i^j \end{cases} \qquad \text{for } j = 1$$

Eq. 6-4
$$\mathrm{B}_i^j = \begin{cases} 0 & x_i < a_i^j \\ \dfrac{(x_i - a_i^j)}{(b_i^j - a_i^j)} & a_i^j \le x_i \le b_i^j \\ 1 & x_i > b_i^j \end{cases} \qquad \text{for } j = 5$$

Eq. 6-5
$$\mathrm{B}_i^j = \begin{cases} 0 & x_i < a_i^j \\ \dfrac{(x_i - a_i^j)}{(b_i^j - a_i^j)} & a_i^j \le x_i \le b_i^j \\ \dfrac{(x_i - c_i^j)}{(b_i^j - c_i^j)} & b_i^j < x_i \le c_i^j \\ 0 & x_i > c_i^j \end{cases} \qquad \text{for } j = 2,3,4$$

where i =1,2 and j = 1,..,5 .

The crisp values of the input variables are mapped onto the fuzzy plane using *Eq. 6-3* to *Eq. 6-5*. It gives each input variable a membership function relating to the fuzzy sets, B_i^1 to B_i^5. It has to be pointed out that in these equations, B_i^j is used to denote the *membership function*. The reader should be aware that in this thesis, the symbol B_i^j is used for both, the linguistic value as well as the membership function. Strictly speaking (see *Section 6.1.4*), the linguistic value should be denoted using B_i^j, while the membership function using $\mu_{Bi,j}(x_i)$.

In this thesis, 'μ' and '(x_i)' are dropped from the denotation of membership function, thus,

$$\mu_{Bi,j}(x_i) = B_i^j = 0.5$$

is used to indicate that x_i belongs to the linguistic value B_i^j by a membership function of the value 0.5.

The universe of discourse of the output variable is divided into seven linguistic values as shown in *Figure 6-8*. The membership functions of the output values are intentionally made to be symmetrical as this will simplify the defuzzification computation. E^1 to E^7 are the mean of each function and act as the weightings to the weighted average method of defuzzification.

6.3.5 Fuzzy Rule Base

Because each input variable can take any of the 5 linguistic values, 25 (=5x5) rules are formulated. The rules have the typical fuzzy rule structure, using linguistic *variables* in both the antecedent and consequent, and are expressed in IF-THEN manner. They map the input states onto 25 output conditions (C_1 to C_{25}).

The fuzzy rules have the general form,

Eq. 6-6 R^k : IF x_1 is A_1^k AND x_2 is A_2^k, THEN y is C^k.

where

R^k (k = 1, 2, …, 25) is the k^{th} rule of the fuzzy system,

x_1, x_2 are the input linguistic *variables*,

y is the output linguistic *variable*,

A_i^k (i = 1, 2 ; k = 1, 2, …, 25) is the k^{th} fuzzy set defined in the i^{th} input space and A_i^k can take any linguistic *value* associated with x_i ,

C^k is the output condition inferred by the k^{th} rule.

If the denotation is such that the linguistic variables:

x_i for i = 1,2

have the following linguistic values:

B_i^j for i = 1,2 ; j = 1 to 5,

then the rule base can be represented by a Fuzzy Associative Memory (FAM) table as shown in *Table 6-1*.

Table 6-1 *Fuzzy Associative Memory Table.*

x_1 \ x_2	B_2^1	B_2^2	B_2^3	B_2^4	B_2^5
B_1^1	C^1	C^2	C^3	C^4	C^5
B_1^2	C^6	C^7	C^8	C^9	C^{10}
B_1^3	C^{11}	C^{12}	C^{13}	C^{14}	C^{15}
B_1^4	C^{16}	C^{17}	C^{18}	C^{19}	C^{20}
B_1^5	C^{21}	C^{22}	C^{23}	C^{24}	C^{25}

6.3.6 Inference Engine

The FLC design in this project incorporates Mamdani's implication method of inference, which is one of the most popular method in fuzzy control applications. In essence, Mamdani's implication for the fuzzy rule of *Eq. 6-6* is given by

Eq. 6-7 $$\mu_C(y) = \max_k \left[\min[\mu_{A_1^k}(x_1), \mu_{A_2^k}(x_2)] \right] \quad k = 1, 2, \ldots 25$$

The implication has a simple min-max structure which makes it easy to incorporate into hardware. The block diagram in *Figure 6-9* provides an overview of the controller's internal structure. Two input variables are fuzzified, producing the corresponding linguistic values and membership functions (B_i^j). The first phase of Mamdani's implication involves *min*-operation since the antecedent pairs in the rule structure are connected by a logical 'AND'. All the rules are then aggregated using a *max*-operation.

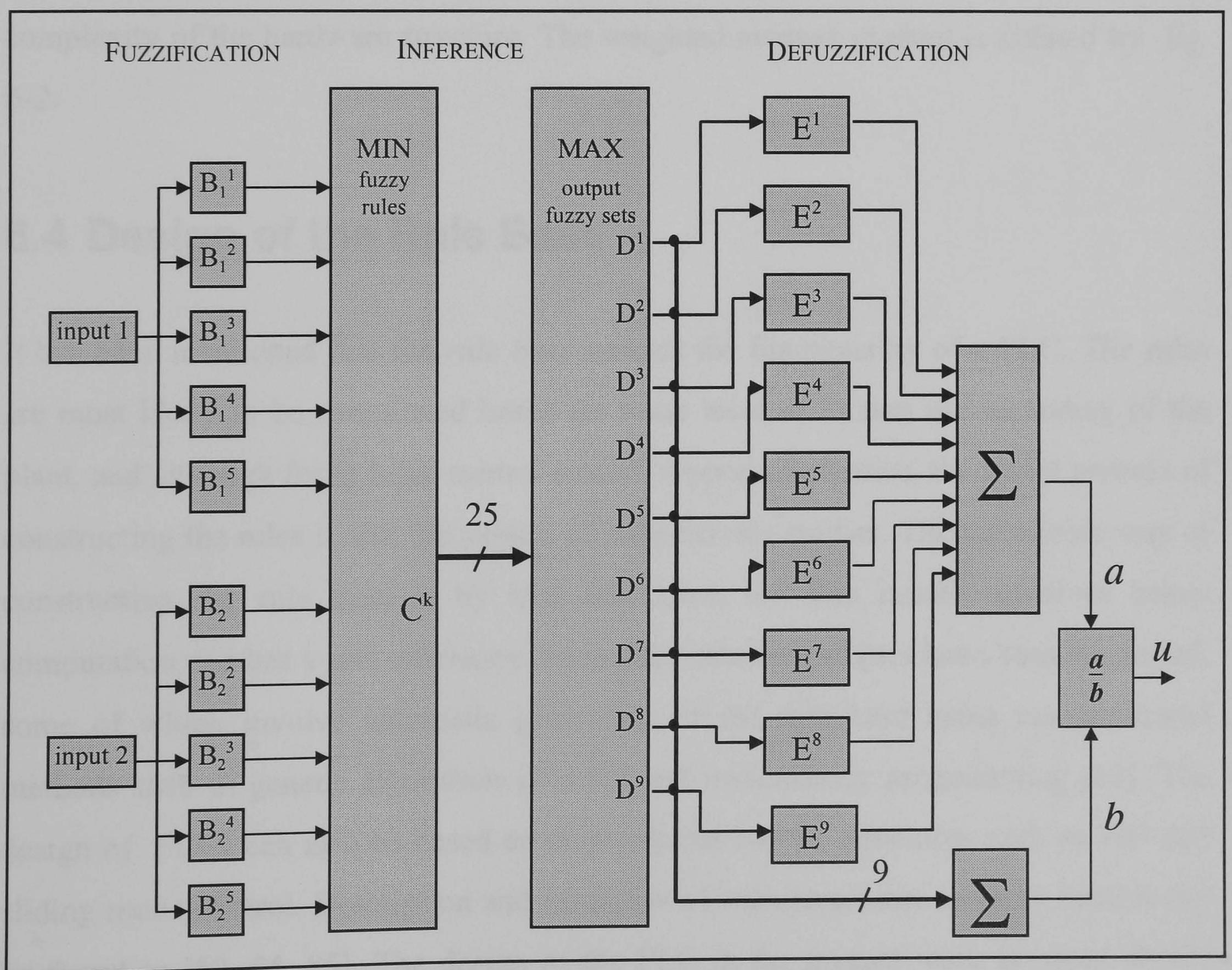


Figure 6-9 *Block diagram of the operations in a fuzzy logic controller.*

6.3.7 Defuzzification Technique

Different defuzzification techniques have different levels of complexities. There have been several studies into the methodology to guide the selection of defuzzification techniques based on the criteria of the design[78,79,80]. The dominant criteria in this design lies in the implementation stage. In order to implement the system into an FPGA, it would be advantageous for the defuzzification technique to be fairly straight forward and to not involve a large number of complex calculations. The weighted average method is viewed to be an appropriate technique for systems involving hardware implementation. Due to the fact that the output membership functions are symmetrical in nature, the mean of the fuzzy sets can be used as weightings for the defuzzification process. This technique requires several multiply-by-a-constant operations and only one division process. The fact that the multipliers have constant values further reduces the complexity of the hardware structure. The weighted average method is defined by *Eq. 6-2*.

6.4 Design of the Rule Base

It has been mentioned that the *rule base* moulds the functionality of a FLC. The rules are most likely to be formulated based on some level of human understanding of the plant, and although fuzzy logic control system supports heuristics, the actual process of constructing the rules is still the subject of considerable studies. The most basic way of constructing the rule base is by trial and error, but this usually involves heavy computation and has a low efficiency. Numerous other techniques have been suggested, some of which involve automatic generation of the rule base using computational methods such as genetic algorithms [81, 82] and evolutionary programming [83]. The design of FLCs can also be based on conventional control structures such as PID and sliding mode control. Description and examples of such structures in fuzzy control can be found in [69, 84, 85]. The design of the FLC in the present work is based on PI-controllers because it is highly suitable for the governing (of the d.c. voltage) system required. The rule base is constructed from the control law of a PI-system.

6.4.1 PI-Control

The Proportional-Integral (PI) controller is a well known system in control engineering. It is, in essence, a lag compensator characterised by the transfer function

Eq. 6-8 $$G(s) = K\left(1 + \frac{1}{T \cdot s}\right)$$

where

G(s) is the gain,

K is the control parameter and

T is the time constant.

The control law is given by the equation

Eq. 6-9 $$u_{PI} = K_p \cdot e + K_I \cdot \frac{1}{T}\int_0^t e \cdot dt$$

where

u is the control signal and

e is the error, given by e = (input value) - (reference value).

Differentiating *Eq. 6-9* gives

Eq. 6-10 $$\frac{du}{dt} = K_P \cdot \frac{de}{dt} + K_I \cdot e$$

In discrete-time system, *Eq. 6-10* can be written as

$$u(kT) - u(kT - T) = K_P \cdot \{e(kT) - e(kT - T)\} + K_I \cdot e(kT)$$

Eq. 6-11 $$\Delta u = K_P \cdot \Delta e + K_I \cdot e$$

where

Δu is the change in u over one sampling period and

Δe is the change in e over one sampling period.

The characteristic of a PI-controller can be represented by the phase plane diagram shown in *Figure 6-10*. A diagonal line where $\Delta u = 0$ divides the control area where Δu is positive and Δu is negative.

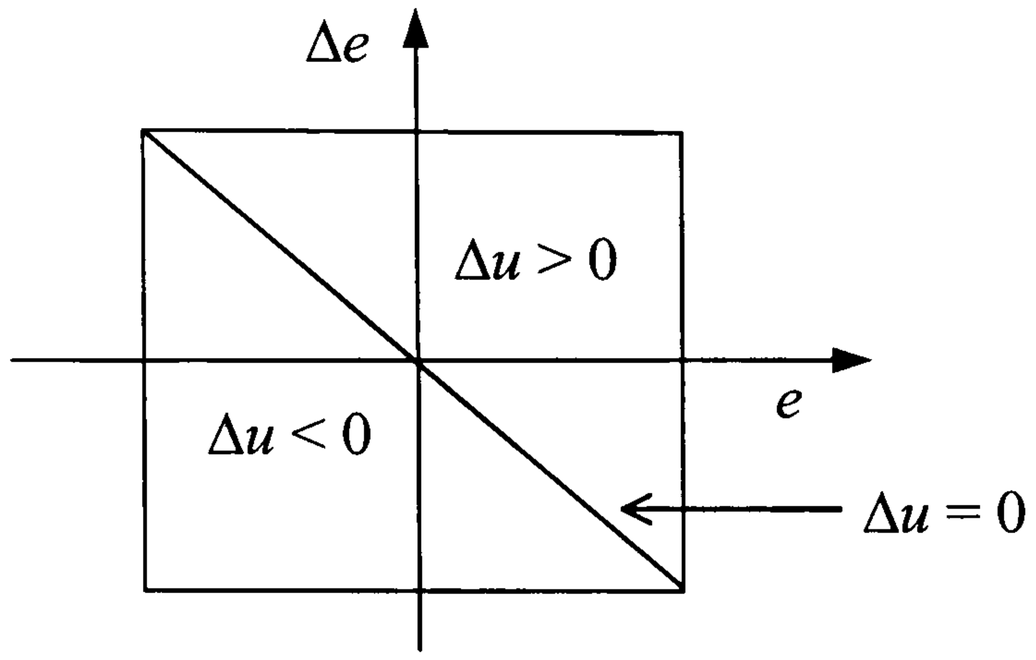


Figure 6-10 *Characteristic of PI-controller.*

6.4.2 PI-like Fuzzy Control

At this stage, the control law in *Eq. 6-11* is not in fuzzy terms. In order to design a fuzzy controller based on the PI-control structure, the following definitions are made:

Let

E be the linguistic variable for the error e,

ΔE be the linguistic variable for the change of error Δe and

U be the linguistic variable for the control output u.

Based on the conventions in *Section 6.1.4*, the following sets are defined:

$\underline{\text{LE}}$ = { Negative Big, Negative, Zero, Positive, Positive Big }

$\underline{\text{L}\Delta\text{E}}$ = { Negative Big, Negative, Zero, Positive, Positive Big }

$\underline{\text{LU}}$ = { Negative Big, Negative, Negative Small, Zero, Positive Small, Positive, Positive Big }

The corresponding PI-control law in IF-THEN rules has the form:

Eq. 6-12 R^k : IF E is A_1^k and ΔE is A_2^k, THEN U is C^k

where

A_1^k can take any linguistic value in the set $\underline{\text{LE}}$,

A_2^k can take any linguistic value in the set $\underline{L\Delta E}$ and

C^k can take any linguistic value in the set $\underline{LU}$.

To implement this design into the FLC, let:

- $x_1 = E$,
- $x_2 = \Delta E$,
- $\{B_i^1, B_i^2, B_i^3, B_i^4, B_i^5\}$ = {Negative Big, Negative, Zero, Positive, Positive Big}

for i =1,2

- $\{D^1, D^2, D^3, D^4, D^5, D^6, D^7, D^8, D^9\}$ = { Negative Very Big, Negative Big, Negative, Negative Small, Zero, Positive Small, Positive, Positive Big, Positive Very Big }

Table 6-2 shows the FAM table of the design.

Table 6-2 *FAM Table for FLC design.*

X_1 \ X_2	**NB**	**N**	**Z**	**P**	**PB**
NB	R^1 u = NVB	R^2 u = NB	R^3 u = N	R^4 u = NS	R^5 u = Z
N	R^6 u = NB	R^7 u = N	R^8 u = NS	R^9 u = Z	R^{10} u = PS
Z	R^{11} u = N	R^{12} u = NS	R^{13} u = Z	R^{14} u = PS	R^{15} u = P
P	R^{16} u = NS	R^{17} u = Z	R^{18} u = PS	R^{19} u = P	R^{20} u = PB
PB	R^{21} u = Z	R^{22} u = PS	R^{23} u = P	R^{24} u = PB	R^{25} u =PVB

NVB Negative Very Big
NB Negative Big
N Negative
NS Negative Small
Z Zero
PS Positive Small
P Positive
PB Positive Big
PVB Positive Very Big

6.4.3 Interfacing Blocks

Figure 6-11 shows a block diagram demonstrating the implementation of the FLC in a stand alone generator system. The control plant in the diagram represents the engine, generator and rectifier system. The notation 'z^{-1}' is used to mark a delay in the signal by one sampling period (the subject of *Z*-transform can be found in [86]).

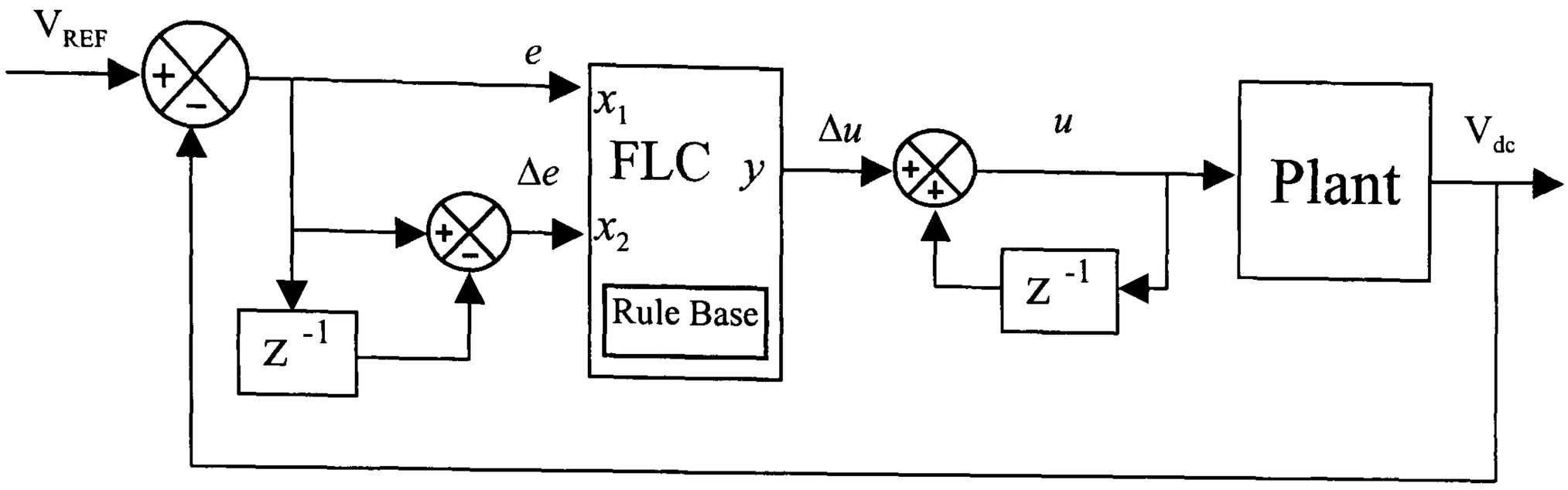


Figure 6-11 *Block diagram of the control system.*

In this application, the input interface converts the d.c. voltage at the output of the plant into *error* and *change of error* which are used as the two inputs to the FLC. Another interface converts the output into the required value for the plant.

The characteristics of the interfacing blocks can be described by the following equations:

Input interface:

$$e = V_{REF} - V_{DC}$$

$$x_1 = e$$

$$x_2 = x_1 - x_1 \cdot z^{-1}$$

Output interface:

$$\Delta u = y$$

$$u = \Delta u + u \cdot z^{-1}$$

Once the design issues of the FLC have been resolved, the next step is to consider the implementation scheme. In this project, the design is achieved using VHDL.

6.5 VHDL Description

The FLC is designed and modelled in an EDA environment using VHDL. The controller is broken-down into 'components', with each component performing a specific function such as fuzzification, inference, etc. This means that the design of each

component can be modified or simply combined with one another to form a complete system. Two levels of design descriptions are presented in this thesis. A behavioural level description is used to test the functional validity of the design. At this level, the design is relatively generic and not restricted by any particular technological constraints. Once the functionality of the design is verified, it is subsequently modified for implementation into the chosen technology. This involves two main tasks: converting the code into structural level and optimising the design. The amount of optimisation required will depend on the target technology. In a general sense, the smaller the targeted device the greater the amount of effort required. In this thesis, the target device is Xilinx XC4010XL FPGA which has a maximum equivalent logic gates of 10,000. This is comparatively small as FPGAs with more than 250,000 gates are available on the market [87].

The FLC is divided into 5 VHDL components. *Figure 6-12* shows a diagram of the FLC architecture. Each component is depicted with its VHDL code file '<filename>.vhd'. The functionality of components **Interface1**, **Interface2** and part of the component **Infer** will determine the characteristics of the FLC. Input and output variables are designed with a resolution of 8 bits.

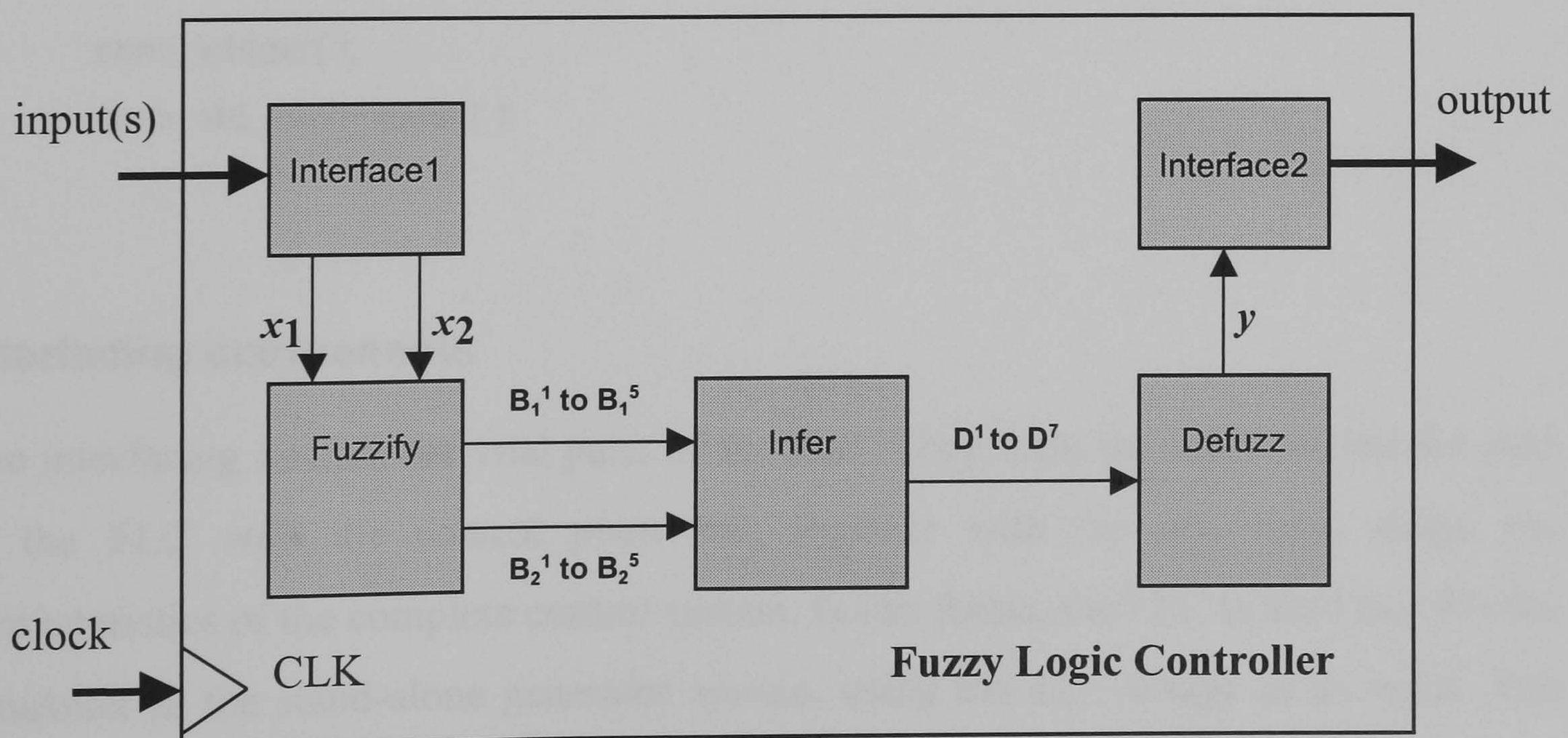


Figure 6-12 *Components in the Fuzzy Logic Controller.*

The ports and signals can be declared either as type **integer** or **std_logic_vector**. An example of the two ways of declaration mentioned here are:

```
port(
    x1 :in integer ;
    x2 :in integer ;
    y  :out integer );
```

and

```
port(
    x1 :in std_logic_vector(7 downto 0);
    x2 :in std_logic_vector(7 downto 0);
    y  :out std_logic_vector(7 downto 0));
```

The choice of type will depend on whether it is easier to write the VHDL code using **integer** or **std_logic_vector**. At *synthesis* level there is little difference between the two types. However, when developing the design for hardware implementation, the *range* of the **integer** must be specified during declaration, a point that is further explained in the next chapter. In this thesis both types are used, depending on the conditions. Conversion functions exist in VHDL to convert between the types [88]. They are:

```
conv_integer( );
conv_std_logic_vector( );
```

Interfacing components

The interfacing circuits are vital parts of the FLC. They form the link between the core of the FLC with the control plant and, together with the rule base, shape the characteristics of the complete control system. In this thesis, the FLC is used as a PI-like controller in the stand-alone generator system, using the d.c. voltage as an input. The transformation required has already been discussed in *Section 6.4.3*.

Components **Interface1** and **Interface2** are used to implement the input and output interfacing functions. This is achieved with the following code:

Part of the code in the VHDL file **Interface1.vhd** is shown below (the complete contents of the VHDL files can be found in *Appendix A*):

```
process(CLK)
    ...
begin
    ...
    PAST_VAR := NOW_VAR ;
    NOW_VAR := error ;
    ...
    DIFF := NOW_VAR - PAST_VAR ;
    -- Normalised input values:
    x1 <= NOW_VAR ;
    x2 <= DIFF * GAIN ;
    ...
end process;
```

In the final version of the design, **Interface2** is embedded within the code for **Defuzz**. The following is a section implements the output interface function:

```
--Output interface
...
U_var := Uz + Y ;
Uz := U_var ;
...
U <= U_var ;
...
```

Fuzzify

The function of the component **Fuzzify** is to convert the crisp input variables into fuzzy values. The membership functions of the input variables are defined in the 'constant-declaration' part of **Fuzzify** as shown below:

```
architecture Fuzzify_arc of Fuzzify is
    constant a1  :integer := - 60;
    constant b1  :integer := - 30;
    constant a2  :integer := - 10;
    constant b2  :integer := - 30;
    constant c2  :integer := 0;
    ...
```

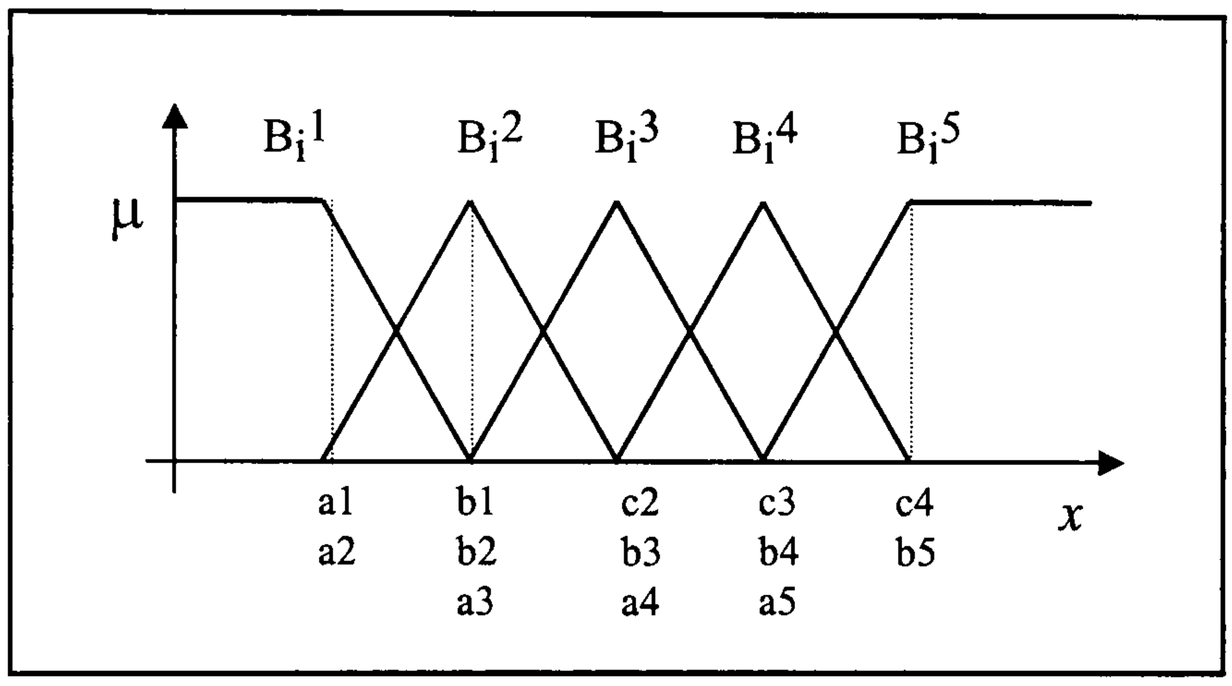


Figure 6-13 *Membership function of input variables.*

The membership values of the input variables to the linguistic values are then assigned based on these constant values. Thus, the membership functions can be slightly modified by redefining the constants. However, the shape of the membership functions would remain more or less the same, i.e. L-, Γ- and Λ-functions as shown in *Figure 6-13*.

Fuzzy Inference

The component **Infer** performs Mamdani's *min-max* inference to obtain the resulting fuzzy output as a consequence of the rule base. A model, as shown in *Figure 6-14*, is developed to represent the inference engine in such a way that it can be easily adopted into the hardware description. The left hand side of the diagram shows the implication of 25 output conditions by the fuzzy rule antecedent using *min*-operators. Each output condition models the consequence of a single rule.

Therefore, the rule

$$R^1 : \text{IF } x_1 \text{ is } B_1^{\,1} \text{ and } x_2 \text{ is } B_2^{\,1}, \text{ THEN U is } C^1$$

becomes

$$C^1 = \min[B_1^{\,1}, B_2^{\,1}] .$$

The *max*-operator is used to take into account the combined effect of all the rules. The 25 output conditions are aggregated into 7 linguistic values (D^1 to D^7) based on the conditions set by the rules. This operation is depicted by the right hand side blocks in *Figure 6-14*.

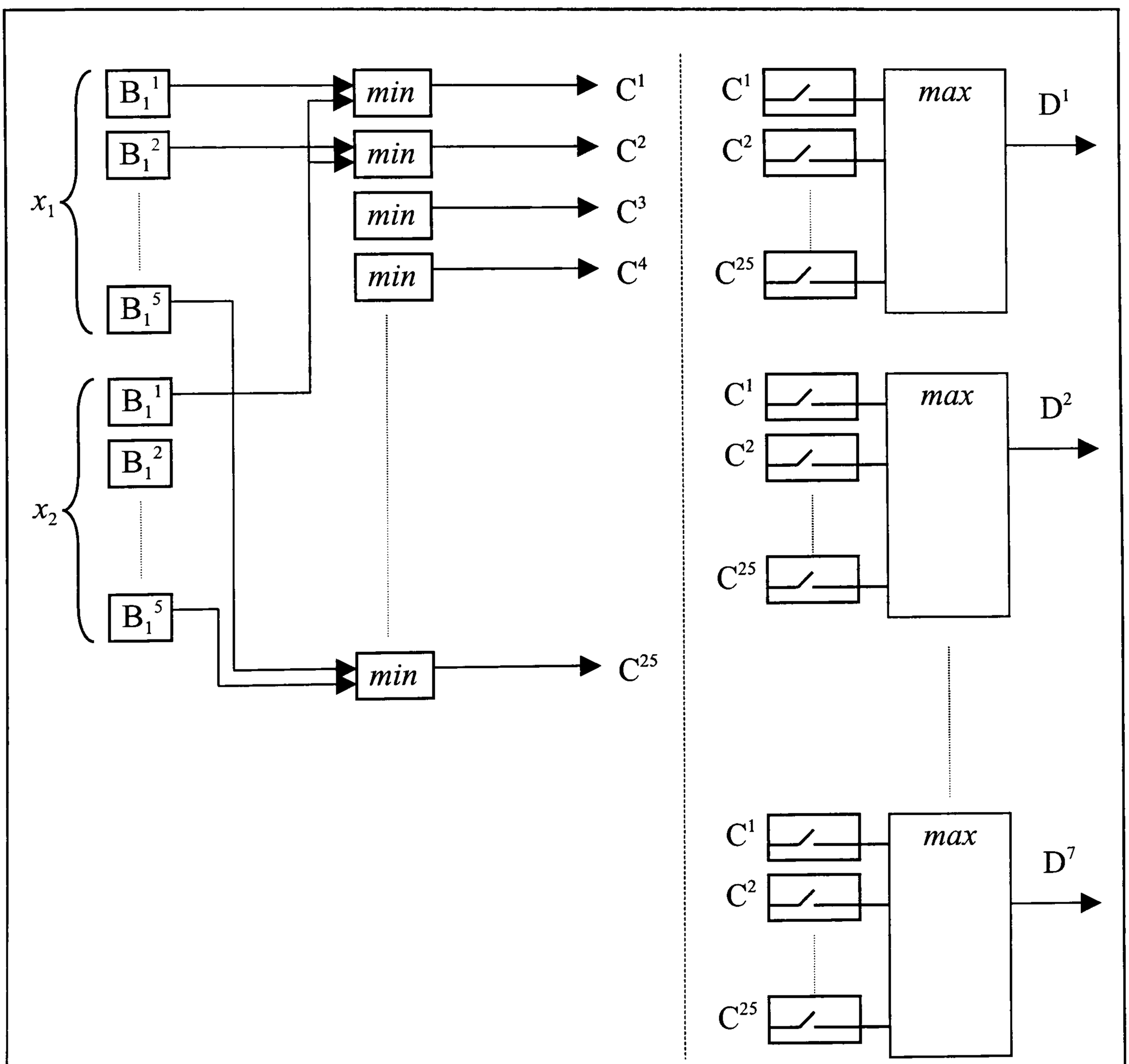


Figure 6-14 *Model of the inference engine.*

For the purpose of illustration, switches are used to model the association of the output conditions to the linguistic values. A switch is set 'ON' if an output condition is associated with the linguistic value for the block of *max*-operation in question. This means that the collective effect of the rule base can be modelled by defining the status of the switches. For example, in the design of the FVG (refer to FAM table in *Table 6-2*), the membership function of the linguistic value D^1 is the aggregate of conditions C^{20}, C^{24} and C^{25}. This is modelled by 'connecting' the three output conditions to the *max*-operation of D^1 and 'disconnecting' the other conditions, as illustrated by the block in *Figure 6-15*. The operation is described by the function:

$$D^1 = \max[C^{20}, C^{24}, C^{25}].$$

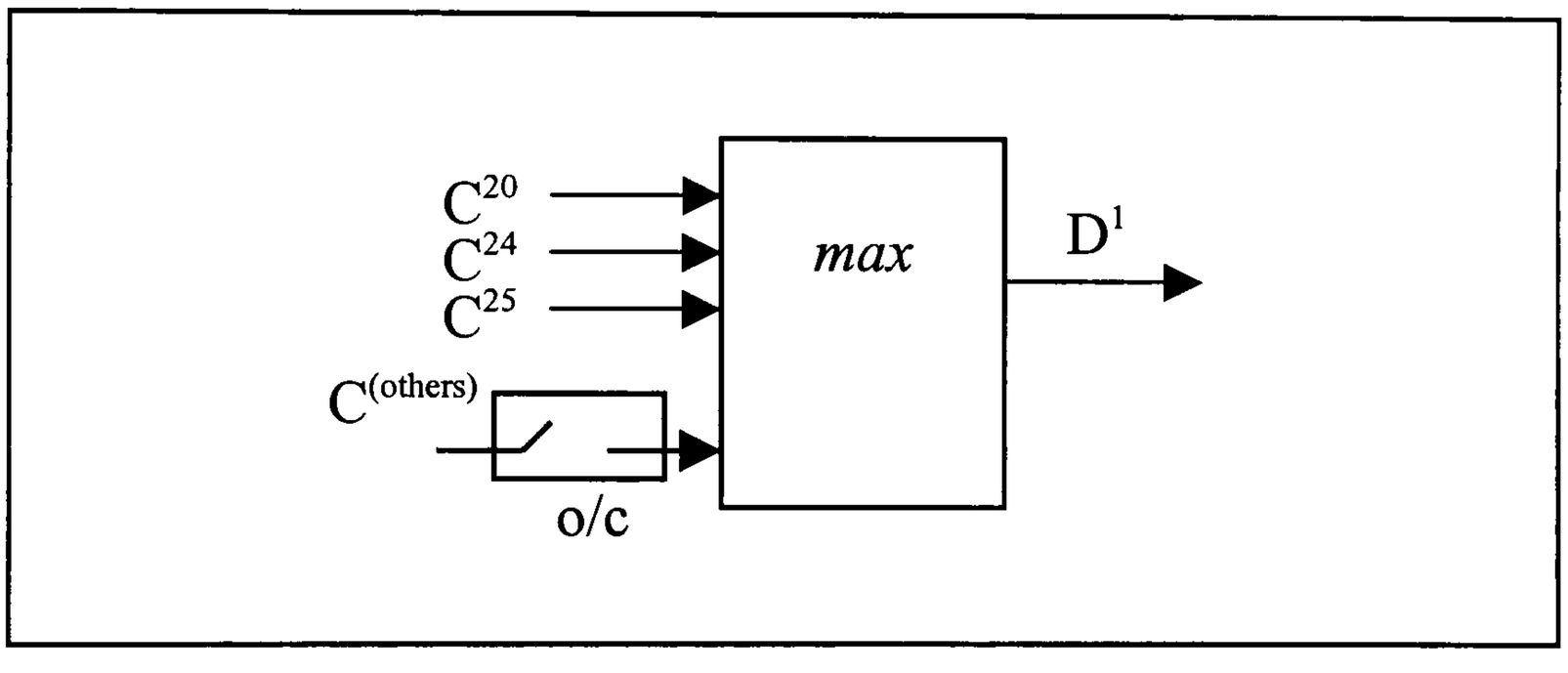


Figure 6-15 *Model of the process of aggregating the fuzzy rules.*

Using the same principle, the entire set of fuzzy rules of the FVG is implemented into the VHDL component **Infer** simply by defining the association between the output conditions and the linguistic values (D^1 to D^7). The following functions are used to incorporate the rule base of the FVG into the FLC.

$D^1 = C^1$

$D^2 = \max[C^2, C^6]$

$D^3 = \max[C^3, C^7, C^{11}]$

$D^4 = \max[C^4, C^8, C^{12}, C^{16}]$

$D^5 = \max[C^5, C^9, C^{13}, C^{17}, C^{21}]$

$D^6 = \max[C^{10}, C^{14}, C^{18}, C^{22}]$

$D^7 = \max[C^{15}, C^{19}, C^{23}]$

$D^8 = \max[C^{20}, C^{24}]$

$D^9 = C^{25}$

Defuzziffier

The function of the component **Defuzz** is to convert the fuzzy output value of the control system into the corresponding crisp value. This is achieved using the weighted average defuzzification method. From *Eq. 6-2* it is evident that this defuzzification operation requires several multipliers and a divider. Behavioural modelling in VHDL supports multiplication and division but these operations are complicated to realise in the synthesis and implementation stages. However, in this chapter only the functional simulation is discussed. Therefore, the multiplication operator '*' and the division operator '/' are used. At subsequent stages, modifications are required as most synthesis tools do not support the division operator '/'. In addition, using a lower level of design description has the advantage of requiring less gates than the '*' operator. Using the multiplication and division operators results in the defuzzification code being fairly straightforward as shown below (the complete code is included in *Appendix A-6*):

```
DEFUZZ_PROCESS:
process(CLK)
    variable Dividend, Divisor : integer;
begin
    if CLK'event and CLK='1' then
        Dividend   := (E1*D1)+(E2*D2)+ (E3*D3)+(E4*D4)+(E5*D5)+ (E6*D6)+(E7*D7)+
                      (E8*D8)+ (E9*D9);
        Divisor    := (D1+D2+D3+D4+D5+D6+D7+D8+D9);
        if Divisor = 0 then
            -- *** Avoid division by zero ***
            Y := 0;
        else
            Y := (Dividend/Divisor);
            ...
        end if;
    end if;
    end process;
```

The **dividend** is obtained from the sum of the product of the linguistic values and their respective weightings while the **divisor** is simply the sum of all the linguistic values. A section of the design, marked "*****Avoid division by zero*****", is dedicated to checking if the value of the **divisor** is zero and taking the appropriate measures to avoid a '*division by zero*' error.

Once all the components of the FLC are successfully designed, a top hierarchy VHDL component, **Control**, is created to bind them together in the manner shown in *Figure 6-12* such as to form the complete controller. The code for this component is included in *Appendix A-7*. Within **Control**, the interfacing components, **Fuzzify**, **Infer** and **Defuzz** are *instantiated* and connected to one another. **Control** also features two clocking signals for synchronising its internal components. By combining the completed design of **Control** with the plant models presented in *Chapter 4*, the performance of the controller can be analysed using computer simulations.

6.6 Simulations

A series of computer simulations are conducted on the FLC to analyse its performance before proceeding to hardware implementation. For this task, a VHDL test component, **Sim**, is created to simulate the control environment. By using the models developed in *Chapter 4*, **Engine** and **Genrect**, it is possible to simulate the relationship between the FLC and the control plant within the VHDL platform in order to study the controller. *Figure 6-16* shows a simplified block diagram of the structural composition of **Sim**. The shaded blocks represent components which have been previously described and are *instantiated* inside the structure of **Sim**. The white blocks are sections of code which perform the data type conversions between the signals of the plant models and those of the controller. A listing of the VHDL code for **Sim** is found in *Appendix A-9*.

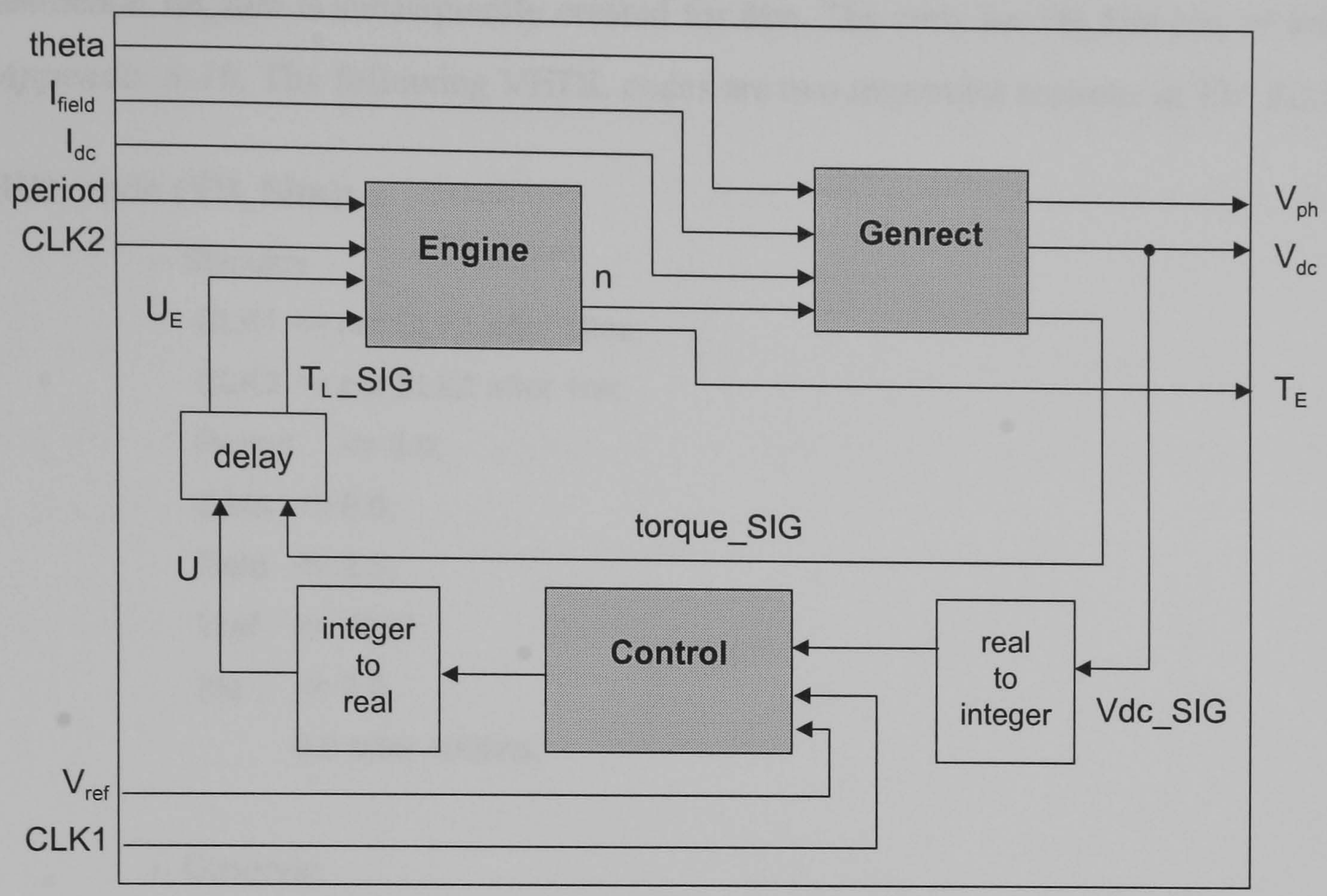


Figure 6-16 *Block diagram of* **Sim**.

The method adopted for running the simulation is by using a VHDL testbench. A testbench is a VHDL design written specifically to provide a test environment for the design under study. A model of the testbench is shown in *Figure 6-17*. It is made up of two main parts, the stimulus and the observer. The former generates the necessary input signals to the design while the observer reads and records the output signals resulting from the stimulus signals.

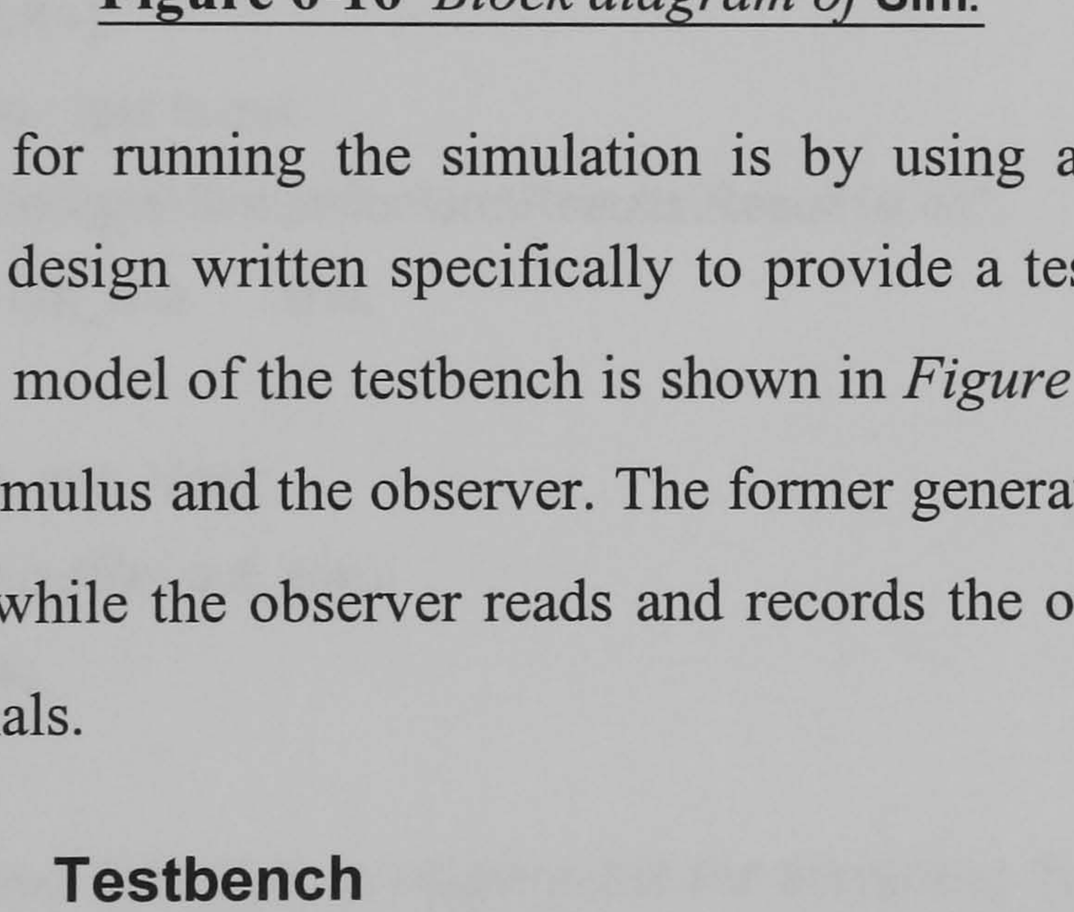


Figure 6-17 *Model of a VHDL testbench.*

A testbench **TB_Sim** is subsequently created for **Sim**. The code for **TB_Sim** can be found in *Appendix A-10*. The following VHDL codes are two important sections in TB_Sim:

VHDL code (TB_Sim):

```
        -- Stimulus
            CLK1 <= not CLK1 after 10ns;
            CLK2 <= not CLK2 after 1ns;
            Period    <= 3.0;
            theta  <= 0.0;
            Ifield  <= 2.5;
            Vref   <= 1000;
            Idc    <= 2.0,
                      6.0 after 4000ns;

    -- Observer
    -- Write results into file
    process (CLK1)
        file outfile : text is out
        "C:\My Designs\Simulation\src\Results\Result1a.txt";
        variable out_line    : line;
    begin
        write(out_line, Vdc);
        writeline(outfile, out_line);
    end process;
```

The first section, marked *Stimulus*, is responsible for assigning the appropriate values to the components in the test environment. In the given example, the load current (measured at the d.c. link) **Idc** is changed from 2.0A to 6.0A when t = 4000ns. The other section of code (with the marking *Observer*) records the response of the system and writes the information into a text file. This is achieved using the **write** and **writeline** statements which are found in the **std.textio** library package. In this chapter, graphical representations of the system responses, designed using MS Excel, are shown instead of the raw numerical data. The voltage values shown in the graphs are normalised to the reference voltage.

6.6.1 Simulation 1

In this simulation, the effects of the weightings $\mathbf{E}^{1\text{-}7}$ for the defuzzification process in the component **Defuzz** are investigated. The values of the weightings in ***Simulation No.1a*** are as follows:

$\mathbf{E^1 = -8\ ;\ E^2 = -6\ ;\ E^3 = -4\ ;\ E^4 = -2\ ;\ E^5 = 0\ ;\ E^6 = 2\ ;\ E^7 = 4\ ;\ E^8 = 6\ ;\ E^9 = 8}$

Using the stimulus signals described by the VHDL code below, the d.c. voltage response of the system is simulated.

VHDL code (TB_Sim):

```
-- Stimulus
    CLK1 <= not CLK1 after 10ns;
    CLK2 <= not CLK2 after 1ns;
    Period    <= 3.0;
    theta  <= 0.0;
    Ifield  <= 2.5;
    Vref   <= 1000;
    Idc    <= 10.0;
```

The reference voltage is set to 1000 as this is a convenient value with which to analyse the performance of the controller. In the graphs shown, *Figure 6-18* to *Figure 6-25,* the values of the d.c. voltage in the graphs are normalised to the reference voltage (i.e. d.c. voltage = Vdc/Vref).

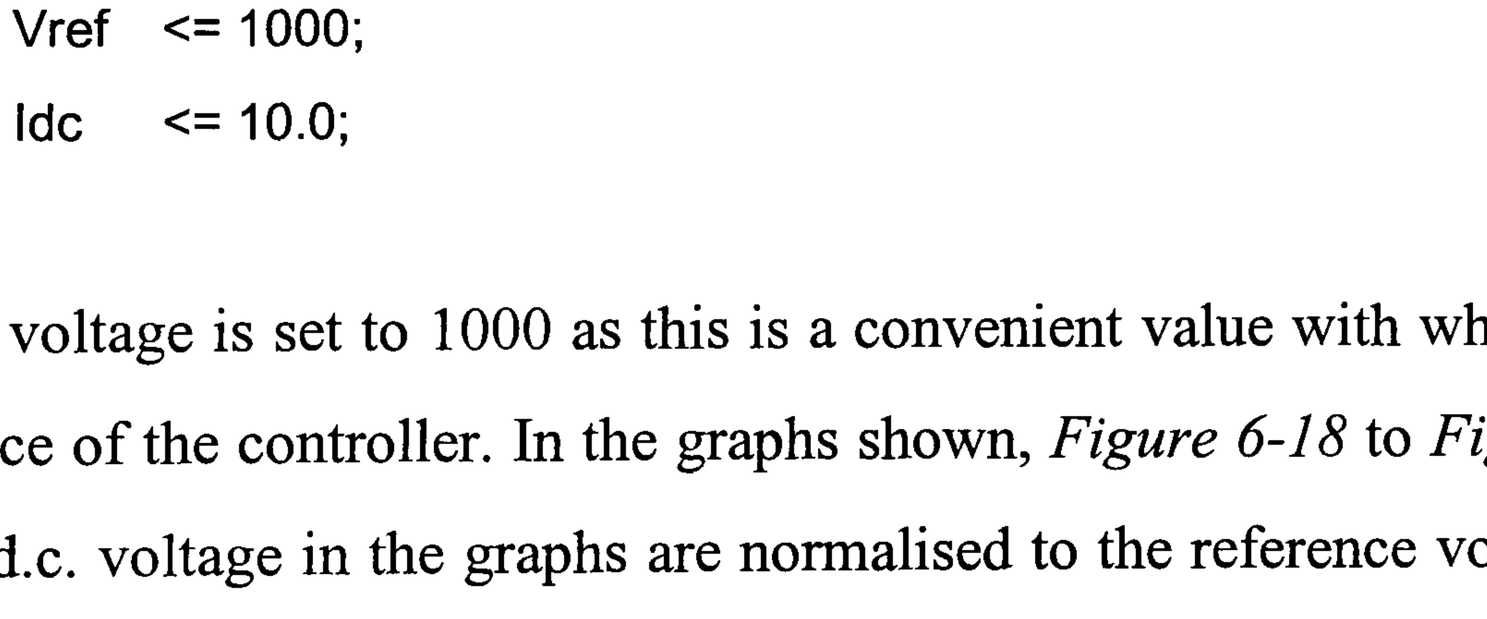


Figure 6-18 *Voltage response in Simulation No.1a.*

It is clearly visible from *Figure 6-18* that the system is unstable. The d.c. voltage oscillates continuously with a magnitude of up to 9% of the reference value. This is partly caused by overcompensation which results from the large weighting values in the defuzzification process. In ***Simulation No.1b***, the weighting values are reduced to:

$$\mathbf{E^1 = -4\ ;\ E^2 = -3\ ;\ E^3 = -2\ ;\ E^4 = -1\ ;\ E^5 = 0\ ;\ E^6 = 1\ ;\ E^7 = 2\ ;\ E^8 = 3\ ;\ E^9 = 4}$$

The voltage response, plotted in the graph in *Figure 6-19*, shows that the system has been stabilised by reducing the weightings in the defuzzification process. However, there is a steady state error of almost 7.5%. In order to improve the performance in this respect, it is necessary examine the shapes of the input membership functions in the component **Fuzzify** and its effect on the control performance.

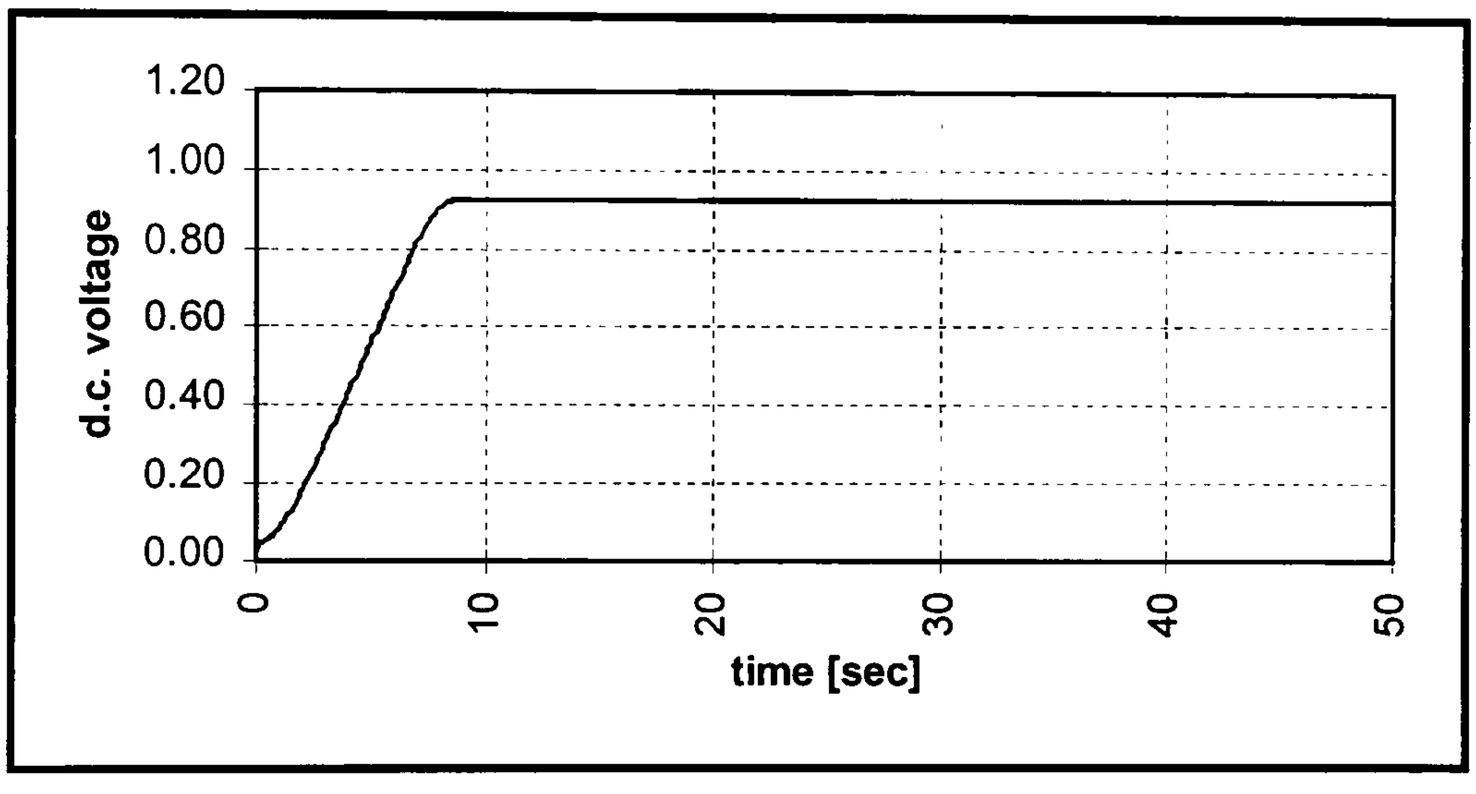


Figure 6-19 *Voltage response in Simulation No.1b.*

6.6.2 Simulation 2

Although not previously mentioned, the conditions set in the fuzzification process in ***Simulations No.1a*** and ***No.1b*** are as follows (refer to *Section 6.5* for the relation between the constants and the membership functions):

VHDL Code (Fuzzify):

Simulation No.1a&b

```
constant a1  :INTEGER   := -80;
constant b1  :INTEGER   := -40;
constant a2  :INTEGER   := -80;
constant b2  :INTEGER   := -40;
```

```
constant c2  :INTEGER   :=   0;
constant a3  :INTEGER   := -40;
constant b3  :INTEGER   :=   0;
constant c3  :INTEGER   :=  40;
constant a4  :INTEGER   :=   0;
constant b4  :INTEGER   :=  40;
constant c4  :INTEGER   :=  80;
constant a5  :INTEGER   :=  40;
constant b5  :INTEGER   :=  80;
```

Plotted on a graph, these conditions present a membership function diagram similar to the one shown in *Figure 6-20*. The shapes of the membership functions of the three linguistic values in the middle are identical, with each of them spanning 80 units over the *x*-axis. By making the linguistic value at the centre (**Zero**) more focused around the value *zero*, it is possible to reduce the magnitude of the steady state error. The FLC design is constructed such that this task can be achieved simply by changing some values of the 13 constants (**a1-4, b1-5, c2-5**) inside **Fuzzify**.

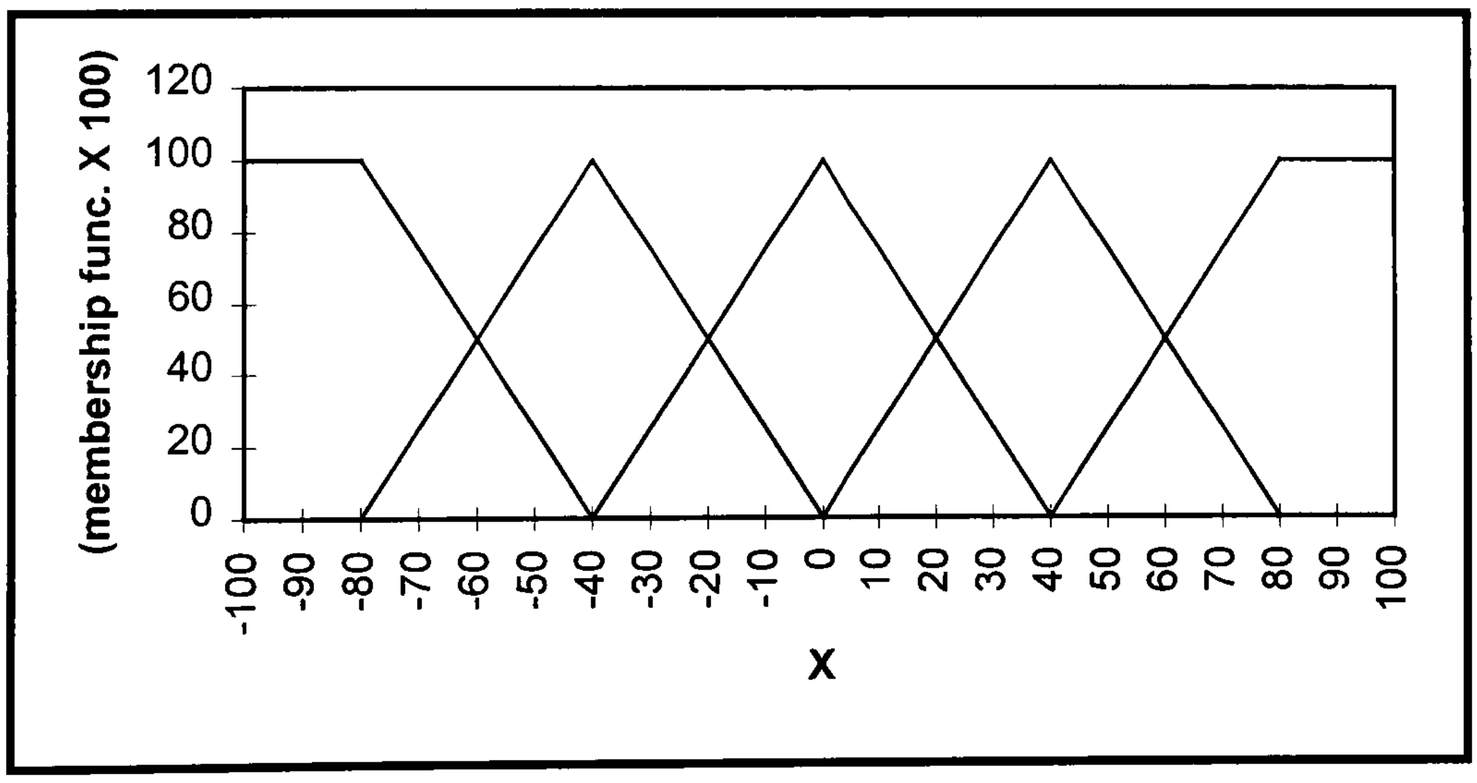


Figure 6-20 *Input variables membership functions in Simulations No.1a and No.1b.*

However, if the membership function is made too narrow, the system loses its stability. This is demonstrated by ***Simulation No.2a***, in which the membership function of the input variables is changed to that of *Figure 6-21* whereby the linguistic value **Zero** ranges between (-5) and (+5). The voltage response is plotted in the graph in *Figure 6-22*. The system response is oscillatory albeit with a smaller magnitude (2.5%) than that of ***Simulation No.1a***.

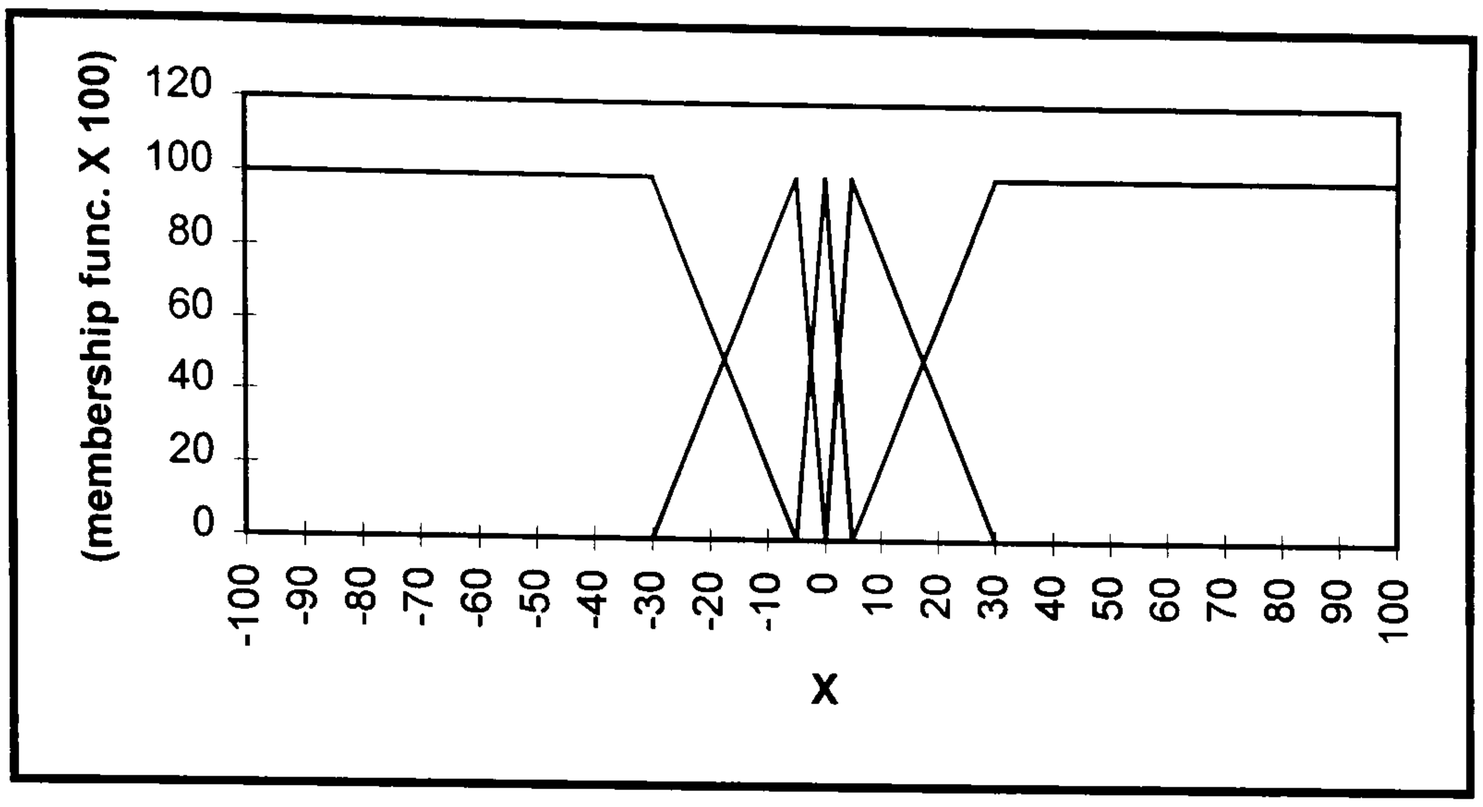


Figure 6-21 *Input variables membership functions in Simulations No.2a.*

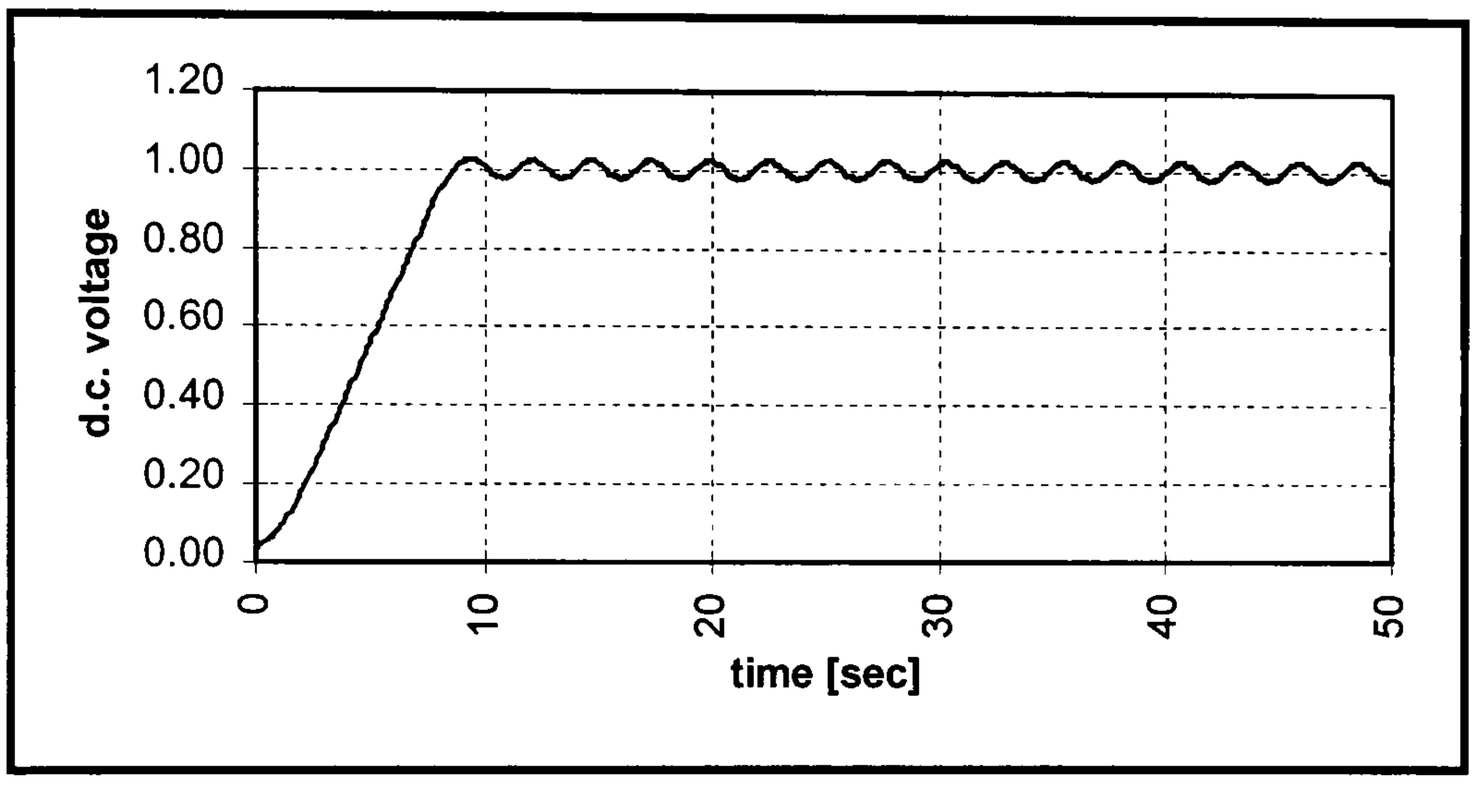


Figure 6-22 *Voltage response in Simulation No.2a.*

In ***Simulation No.2b***, the membership function of the input variables is modified to resemble the diagram in *Figure 6-23*. The result, shown by the graph in *Figure 6-24* clearly proves that, when the correct balance is achieved in the choice of membership function, the FLC performance is considerably improved. The steady state error in this case is less than 1.6%.

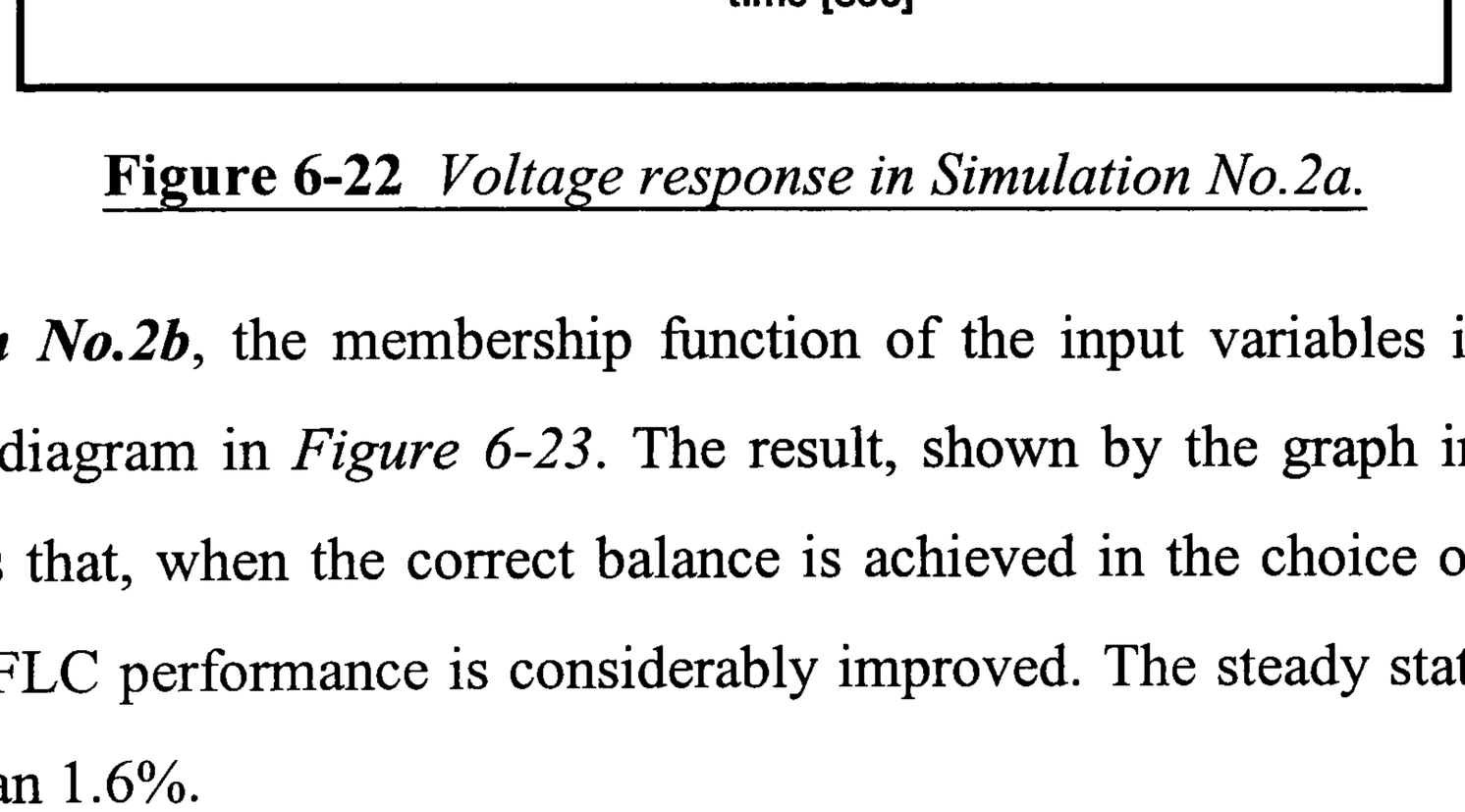


Figure 6-23 *Input variables membership functions in Simulation No.2b.*

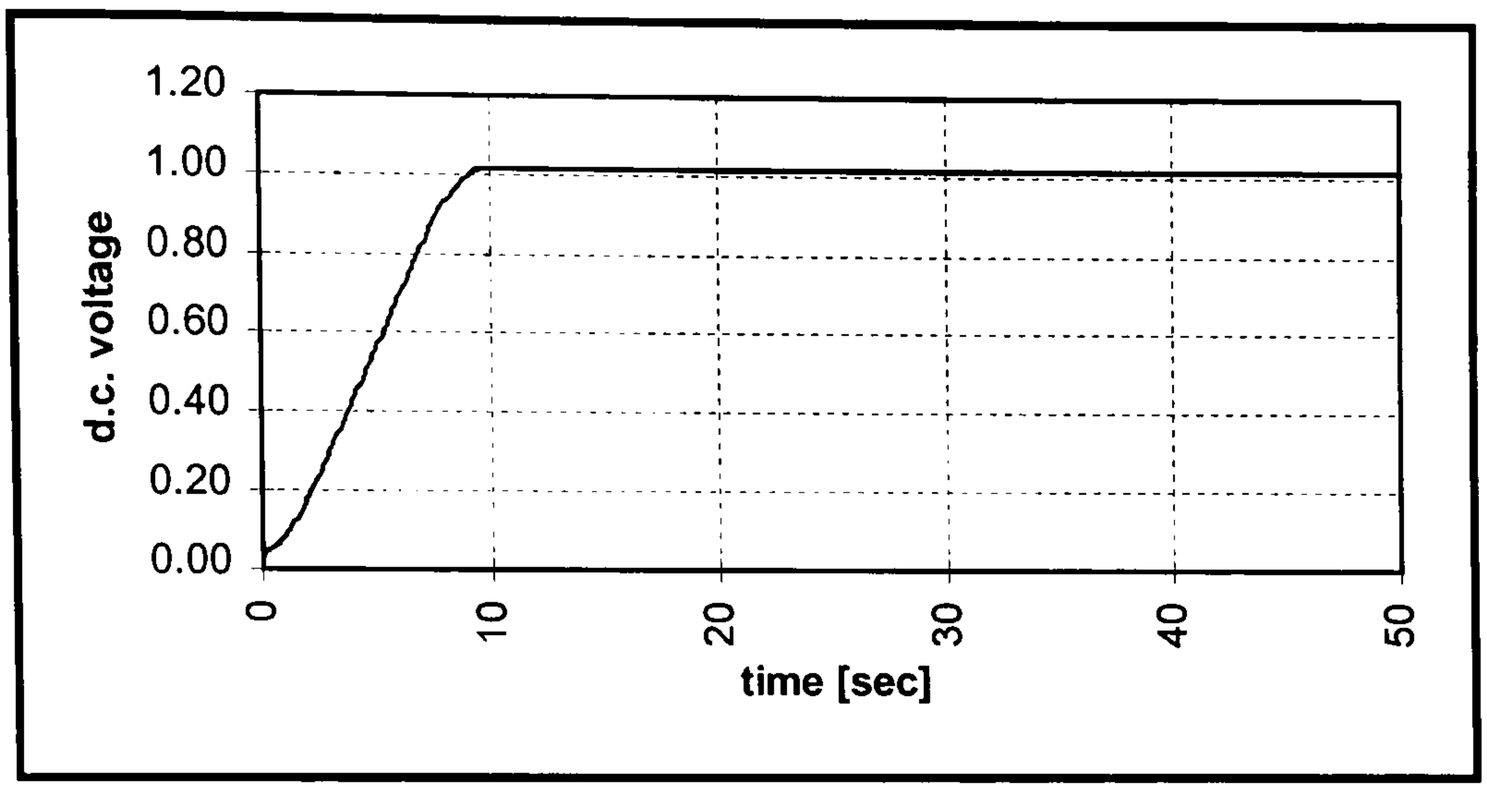


Figure 6-24 *Voltage response in Simulation No.2b.*

6.6.3 Simulation 3

Once the parameters in **Fuzzify** and **Defuzz** are properly tuned, the system can be simulated under varying load conditions. In ***Simulation No.3a***, the d.c. load current is changed from 2A to 6A. *Figure 6-25* shows the voltage response whereby the change occurs when t = 20 seconds. There is a transient dip in the voltage of less than 14% when the load current increases suddenly but the FLC is shown to be capable of restoring the voltage to its original value in less than 5 seconds (50 cycles).

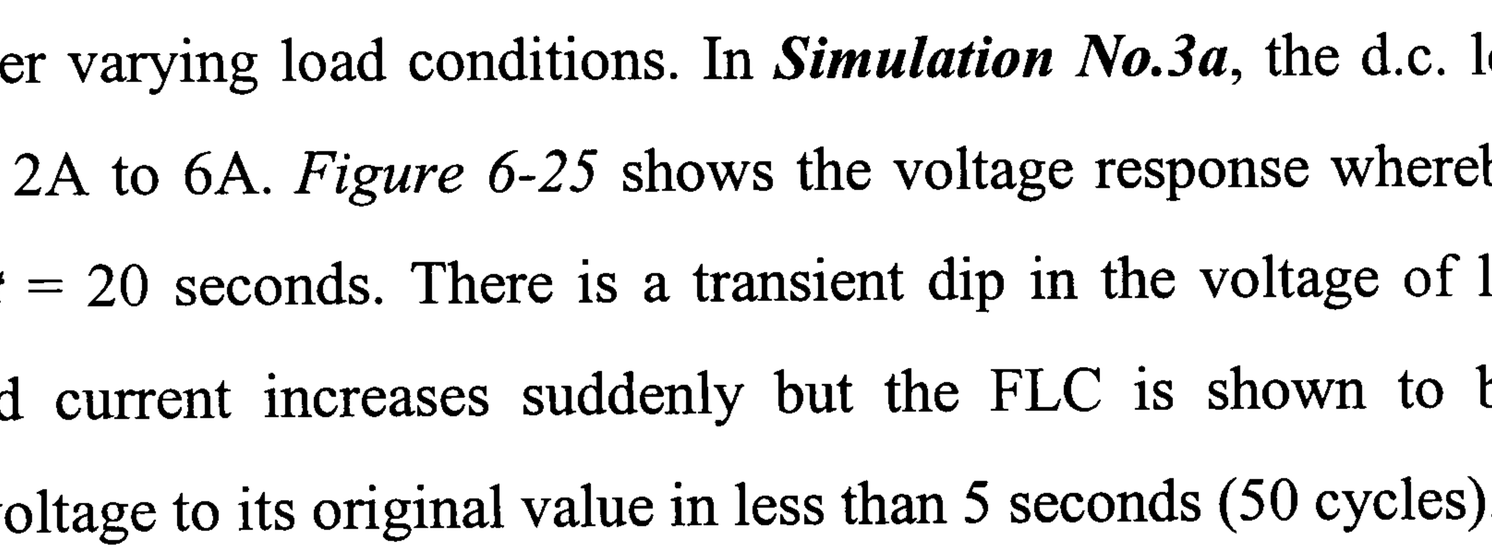


Figure 6-25 *Voltage response in Simulation No.3a.*

This chapter has presented an extensive review of the VHDL design of a Fuzzy Logic Controller, including a brief introduction to the concept of Fuzzy Logic. A series of simulations are presented and the results provided the system validation necessary to proceed to hardware implementation with confidence. In the next chapter, the issues surrounding the implementation of the FLC are studied and a detailed description of the design process involving the use of Field Programmable Gate Arrays (FPGAs) for hardware realisation is given.

Chapter 7

FPGA Implementation

In order to transform a behavioural level VHDL design such as the one presented in the previous chapter into a hardware design, a number of considerations have to be made. To fully appreciate the issues involved, it is necessary to have some understanding of the hardware device's internal architecture. This chapter begins with an introduction to the internal architecture of the Xilinx FPGA used in the project. It also explains the need to optimise the design before implementation and presents the work involved in modifying the design for this purpose.

The *synthesis* and *implementation* processes are discussed in considerable detail. *Synthesis* is the act of converting the VHDL code in to a *netlist.* A netlist is a standard method of describing the design at a level of abstraction that is suitable for hardware implementation, either at an architectural level which consists of logic blocks or at a logic level which is made up of logic primitives. *Implementation* is the process of converting the netlist into a format that can be downloaded into the target technology. This procedure is very technology-specific since different devices require different formats.

7.1 The Xilinx FPGA

The target technology for this design is the XC4010XL-PC84, a member of the Xilinx XC4000XL Series of FPGAs. These devices are fully re-configurable and can be reprogrammed an unlimited number of times. The XC4000XL Series is a family of high

performance 3.3v devices based on SRAM technology. Some features of the devices in the Series include synchronous system clock rates of up to 80MHz, internal speed performance exceeding 150MHz and 12mA sink current per output. *Table 7-1* shows other features of some of the devices in the family [89]. The Xilinx XC4010 has a maximum logic gate count of 10,000. Some of the other devices in the Series are much larger, with equivalent logic gate count of up to 85,000. The relevance of this point will become clear later in the chapter when it is demonstrated that although the original FLC design in this thesis is too large to fit into a Xilinx XC4010, it can be readily implemented into other larger devices.

Table 7-1 *Features of XC4000XL Family.*

Device	Max. Logic Gates	CLB Matrix	Total CLBs	No. of Flip Flops	Max. User I/O
XC4005XL	5 000	14 x 14	196	616	112
XC4010XL	10 000	20 x 20	400	1 120	160
XC4013XL	13 000	24 x 24	576	1 536	192
XC4020XL	20 000	28 x 28	784	2 016	224
XC4044XL	44 000	40 x 40	1 600	3 840	320
XC4085XL	85 000	56 x 56	3 136	7 168	448

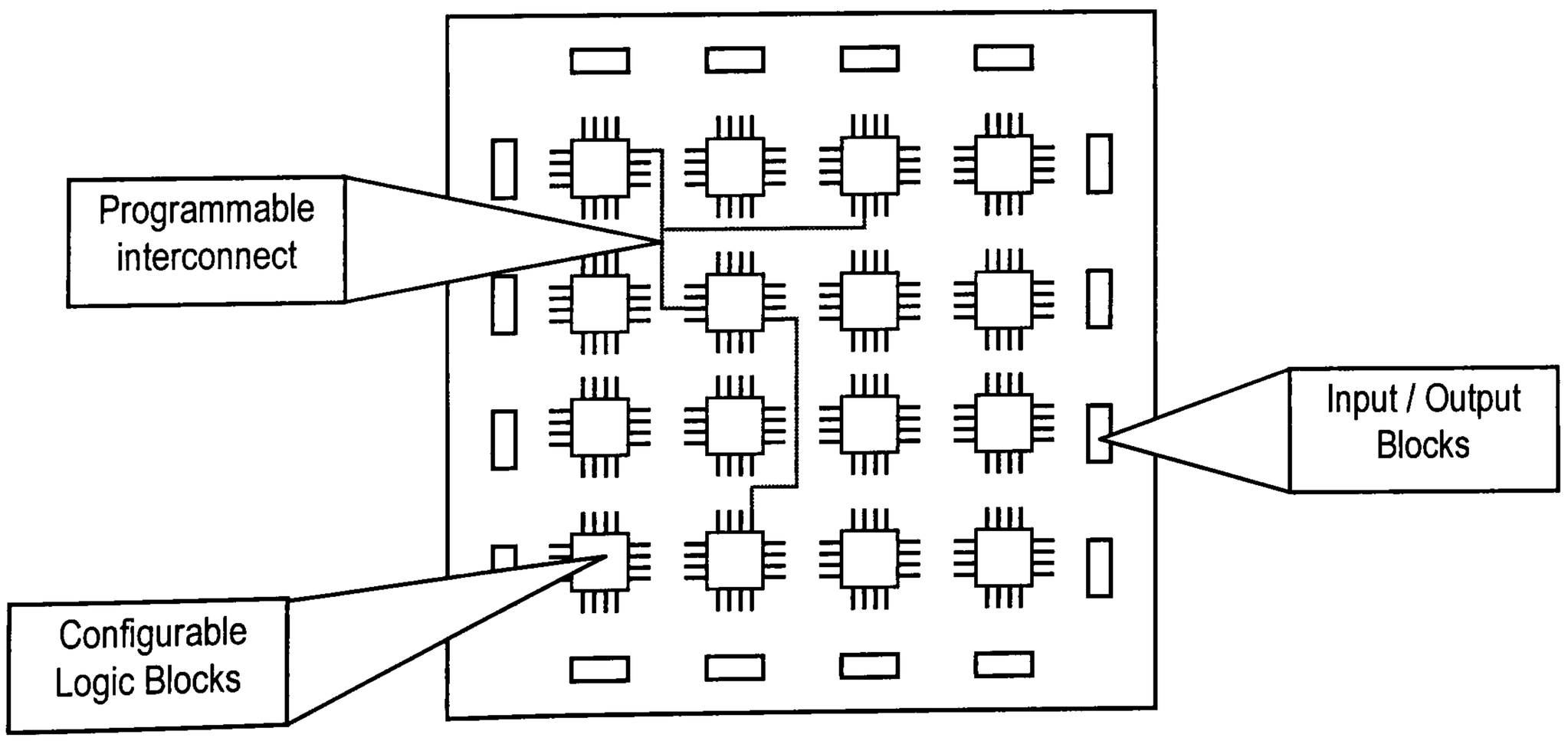


Figure 7-1 *A simplified view of the FPGA architecture.*

A simplified view of the internal architecture of the Xilinx XC4000XL Series FPGA is shown in *Figure 7-1*. The device is mainly made up of a matrix of Configurable Logic Blocks (CLBs), some peripheral Input/Output Blocks (IOBs) and a net of metal interconnects. The logic functions and interconnections are determined by internal static memory cells which can be programmed to customise the functionality of the array.

7.1.1 Configurable Logic Blocks

The CLBs provide the functional elements for implementing most of the desired logic. It contains two 4-input function generators, one 3-input function generator, two flip flops, a few multiplexors and other logic components. The function generators are capable of implementing any arbitrarily defined Boolean operation and can also be used as memory lookup tables. Output(s) can either be passed onto the interconnect network directly or through storage elements such as flip flops. The CLBs are able to implement a variety of logical operations, from *any* function up to 5 variables to *some* functions up to 9 variables. The maximum gate count is based on the equivalent logic gates in the CLBs. It will be demonstrated later in the chapter that it is possible to use up all the CLBs in a device without actually utilising the majority of the logic gates. Efficient use of CLBs improves the overall cell usage as well as the speed performance.

7.1.2 Input / Output Blocks

The user-configurable IOBs link the internal structure of the FPGA to the external package pins. Each IOB can be configured for input, output or bi-directional signal. As inputs, the IOBs can either be programmed as an edge-triggered flip flop or a level-sensitive latch. Despite being 3.3V devices, the I/O terminals on the XC4000XL can tolerate 5V signals making them TTL compatible as well as 5V/3.3V CMOS compatible. As outputs, they are pulled to the 3.3V positive supply and have a guaranteed sink current of 12mA. The high sink current feature means that external buffers are not normally required. By default, the output pull-up structure is configured as a TTL-like totem-pole but they can also be globally configured as CMOS drivers.

7.1.3 Programmable Interconnects

The interconnects can be programmed and reprogrammed such that the blocks are connected to produce the desired effect. The Xilinx XC4000XL FPGAs can be reconfigured an unlimited number of times. The connections are composed of metal segments with programmable switching points and switching matrices.

There are three types of interconnects in these FPGAs :

- CLB routing
- IOB routing which are called VersaRing
- Global routing

Each CLB in the array is surrounded by a net of CLB routing and they are connected to the CLB inputs and outputs. Further information about Xilinx FPGAs can be found in [89].

7.2 Structural VHDL design

The design of the FLC presented in *Chapter 6* did not take into account the *synthesis* and *implementation* considerations and the limitations of the target technology. These considerations have to be addressed when preparing a design for downloading into hardware, usually a PLD, FPGA or ASIC. In the present project, the design is prepared for implementation into a Xilinx XC4010XL FPGA. This device has a total of 400 CLBs and an equivalent gate count of up to 10 000.

Using the EDA tool, *Xilinx Foundation Series F1.5*, a VHDL design can be *synthesised* then *implemented* for a particular device. *Figure 7-2* shows a simplified block diagram of the process. In the synthesis stage, the VHDL code is converted into a netlist, which is a format that contains the structural hardware description of the design. A function is considered to be NOT 'synthesisable' if the synthesis tool cannot readily convert it into hardware description. Functions such as mathematical division and trigonometric operations are not synthesisable using currently available synthesis tools. *Implementation* in this context is the process of converting the netlist into a bitstream

file. The information in the bitstream is used to configure the functionality of the target FPGA. During the implementation stage, the target technology and other hardware design specifications such as pin allocation has to be confirmed. For trouble-shooting and analysis purposes, Xilinx Foundation also generates a status report at each stage of the process.

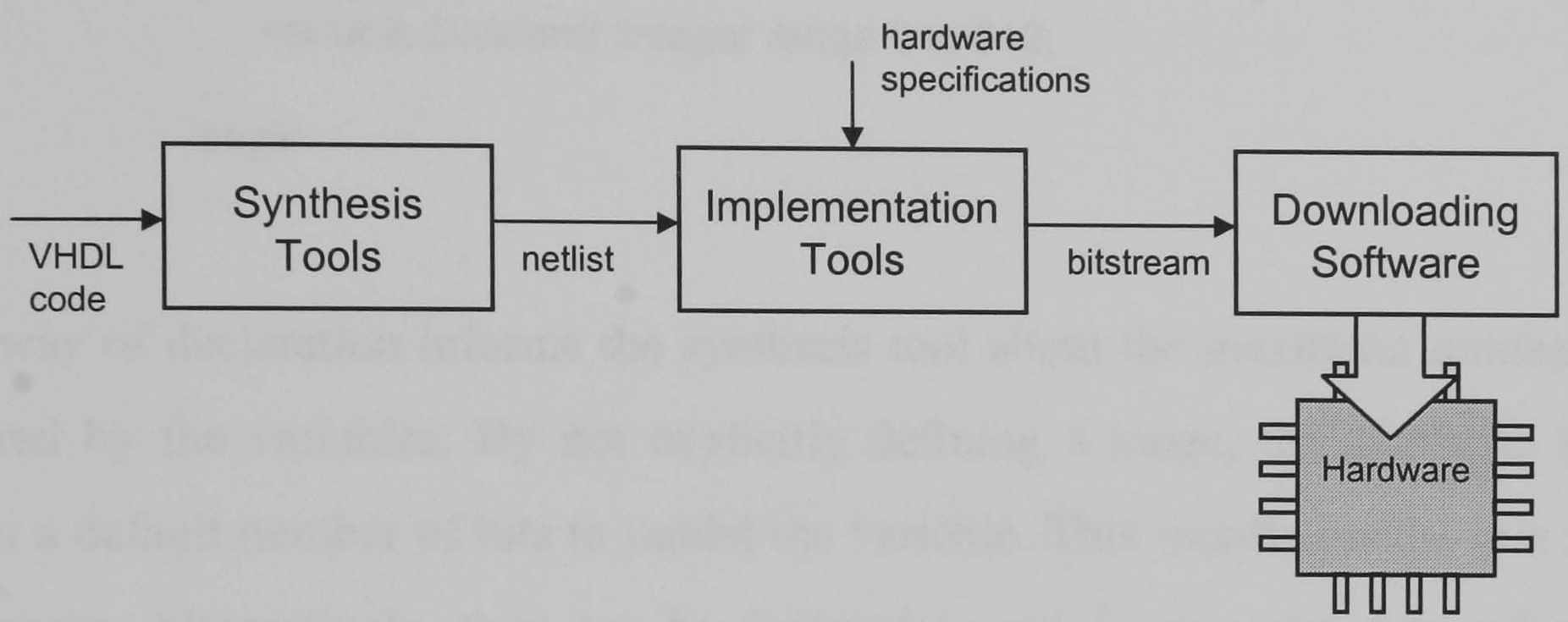


Figure 7-2 *Simplified block diagram of the hardware design process.*

Before a VHDL design can be synthesised, all the VHDL functions which are not 'synthesisable' have to be eliminated and replaced with functions that can be translated into hardware.

The Divider

The FLC's defuzzification process requires a division operation. In the behavioural design, the division is achieved with the following VHDL statements (refer to *Appendix A-6*):

```
process(CLK)
    variable Dividend, Divisor : integer;
begin
    ...
    Y <= (Dividend/Divisor);
    ...
end process;
```

This process is not suitable for synthesis for two reasons:

- The variables **Dividend** and **Divisor** are declared as open-ended integers which have no meaning in hardware design. Integer variables and signals should be given a well defined range such as

```
...
process(CLK)
    variable Dividend :integer range 0 to 512;
...
begin
...
```

This way of declaration informs the synthesis tool about the maximum number of bits required by the variables. By not explicitly defining a range, the synthesis tool will assign a default number of bits to model the variable. This usually results in a waste of resourcess. Alternatively, they can be declared as *std_logic_vector* type which has a more relevant form in hardware terms. The declaration is written as:

```
...
process(CLK)
    variable Dividend :std_logic_vector(8 downto 0);
...
begin
...
```

- It has already been mentioned that the division operator '*/*' is not supported by present synthesis tools, including Xilinx Foundation. To implement this calculation in hardware, it is necessary to design a digital divider at structural level.

Analysis of Binary Integer Division

Before entering a discussion on binary division, it is appropriate to define some of the terms which are used. The *Dividend* (divA) is defined as the number to be divided while the *divisor* (divB) is the number by which the dividend is divided. The division process can then be expressed by the following equation:

$$\frac{\text{divA}}{\text{divB}} = \text{Y} + \frac{\text{R}}{\text{divB}}$$

where Y is the *quotient* and R is the *remainder* whereby R<divB.

In binary notation, the numbers can be written as

$$\text{divA} = 2^{(n-1)} \cdot a_{(n-1)} + 2^{(n-2)} \cdot a_{(n-2)} \ \ldots + 2^1 \cdot a_1 + 2^0 \cdot a_0$$

$$\text{divB} = 2^{(m-1)} \cdot b_{(m-1)} + 2^{(m-2)} \cdot b_{(m-2)} \ \ldots + 2^1 \cdot b_1 + 2^0 \cdot b_0$$

$$Y = 2^{(n-1)} \cdot y_{(n-1)} + 2^{(n-2)} \cdot y_{(n-2)} \ \ldots + 2^1 \cdot y_1 + 2^0 \cdot y_0$$

Using the pencil-and-paper method, a binary division process is performed in a number of steps. In each step, a *partial dividend* {PD} is divided by the divisor such that if *j* is the index for the steps, then,

$$y_j = \frac{\{PD\}_j - R_j}{\text{divB}}$$

and

$$\{PD\}_{(j-1)} = 2R_j + a_{(j-1)}$$

where

$j = (n-1), (n-2) \ldots \quad , 0$

R_j is the *partial remainder* and R_j<divB.

For example, dividing the binary number "1101" by "11":

$$11\overline{)\{1\}101} \quad 0100$$

using the pencil-and-paper method,

when $j = 3$, {PD}={1} $\Rightarrow y_3 = 0, R_3 = 1,$

when $j = 2$, {PD}={11} $\Rightarrow y_2 = 1, R_2 = 0,$

when $j = 1$, {PD}={0} $\Rightarrow y_1 = 0, R_1 = 0,$

when $j = 0$, {PD}={01} $\Rightarrow y_0 = 0, R_0 = 1.$

R_0 is also the true remainder R.

Therefore, the result of the division can be written as:

$$\frac{1101_2}{11_2} = 0100_2 + \frac{1_2}{11_2}$$

or, in decimal notation,

$$\frac{13}{3} = 4 + \frac{1}{3}$$

Binary Integer Division Algorithm

There are several different algorithms which can perform binary integer division. Some algorithms have parallel operations [90] and require only one clock cycle to carry out the calculation while others operate in a sequential manner and may require more clock cycles, such as the one presented later in this chapter. As a general rule, the algorithms with parallel operations are larger in area size than the sequential ones when they are implemented into hardware. The division algorithm used in this thesis is derived from the pencil-and-paper method of binary division [91].

Let

divA	be an *n*-bit register which holds the unsigned dividend,
divB	be an *m*-bit register which holds the unsigned divisor,
B_p	be a signed (*m*+1)-bit register which holds the positive value of B,
B_n	be a signed (*m*+1)-bit register which holds the negative value of B,
A	be an *n*-bit register which holds the partial dividend {PD},
A_s	be a flip-flop appended to the left end of A to store the sign bit and
Z_sZ	be the temporary register used to store

Using the symbol '←' to denote 'is assigned as', the division algorithm used in this thesis can be written as follows:

- Block 1:

  Initialisation:

  clear A;

  $B_p \leftarrow$ positive value of divB;

  $B_n \leftarrow$ negative value of divB (2s complement);

- Block 2:

For $j = (n\text{-}1)$:

shift left (A)(divA);

$Z_jZ \leftarrow (A_sA + B_n)$;

$q_j \leftarrow \text{not}(Z_j)$;

- Block 3

For $j = (n\text{-}2), (n\text{-}3) \ldots, 1, 0$

shift left AP;

if $q_{(j-1)}$ = '0' then:

$Z_sZ \leftarrow (A_sA + B_p)$;

else if $q_{(j-1)}$ = '1' then:

$Z_sZ \leftarrow (A_sA + B_n)$;

end if;

$q_j \leftarrow \text{not}(Z_s)$;

$A_sA \leftarrow Z_sZ$;

A VHDL code is written to implement this algorithm for a 14-bit dividend and a 9-bit divisor. The complete listing of the code is included in *Appendix B-5*. The code defines the dividend as **divA** , the divisor as **divB** and the quotient is defined as **Y**. It is assumed that the effects of the Remainder on the control performance is small, therefore the Remainder can be truncated altogether. Since the algorithm is sequential, the entire operation is carried out in a VHDL *process*. A circuit is also included to avoid a division-by-zero error by clearing **Y** (assign all bits to zero) when **divB** equals to zero.

Using 2s-complement conversion to obtain the negative value of **divB**, the 'Initialisation' procedure in Block 1 is performed by the following section of the code:

```
-- Initialisation
    -- Clear A
A(13 downto 0) :="00000000000000";
    -- Assign divB(+ve) and divB(-ve)
    Bp :="00000"&divB;
    Bn :=not(Bp)+"00000000000001";
```

It can be observed that the variables are treated as binary-valued bits and bytes instead of whole numbers. This is a realistic view of digital circuits and highlights an important difference between hardware description and software programming. The next sequence of commands converts the dividend **divA** into an unsigned value. This is essential because the algorithm is not designed to cope with negative values of the dividend. From the defuzzification process, it is known that the divisor **divB** is always a positive number. Therefore, it follows that the sign status (whether positive or negative) of the quotient **Y** is always the same as the sign status of **divA**. The functions in Blocks 2 and 3 are implemented by introducing the status bits **Load** and **Ready**. '**Load**=1' relates to the condition whereby a new set of dividend and divisor values are loaded into the divider circuit, i.e. when $j=(n-1)$, while '**Ready**=1' is the condition whereby the division calculation for the recent set of values has been completed, that is when $j=0$. The VHDL code also includes the description of the output interface circuit of the controller which implements the following function:

$$u \leftarrow u \cdot z^{-1} + y$$

where

y is the crisp output from the defuzzifier and

u is the actual control signal.

The section of code that describes this circuit is written as follows:

```
...
if SIGN='1' then
    ...
    - - Negative
    U-var := U_Past - Y;
    ...
else
    - - Positive
    U_var := U_Past + Y;
    ...
end if;
...
```

In any hardware code which involves continuous increment or decrement of numerical values, there is a danger of *overflow*. Therefore the upper and lower limit of the output is set at '11111111' and '00000000' respectively. This prevents the 8-bit output register from overflowing into a 9-bit value (e.g. 11111111 + 1 = [1]00000000).

The VHDL code is subsequently configured as a component and a netlist is created. *Figure 7-3* shows the symbol of the component **Divider** as viewed in the Xilinx Foundation Schematic Editor.

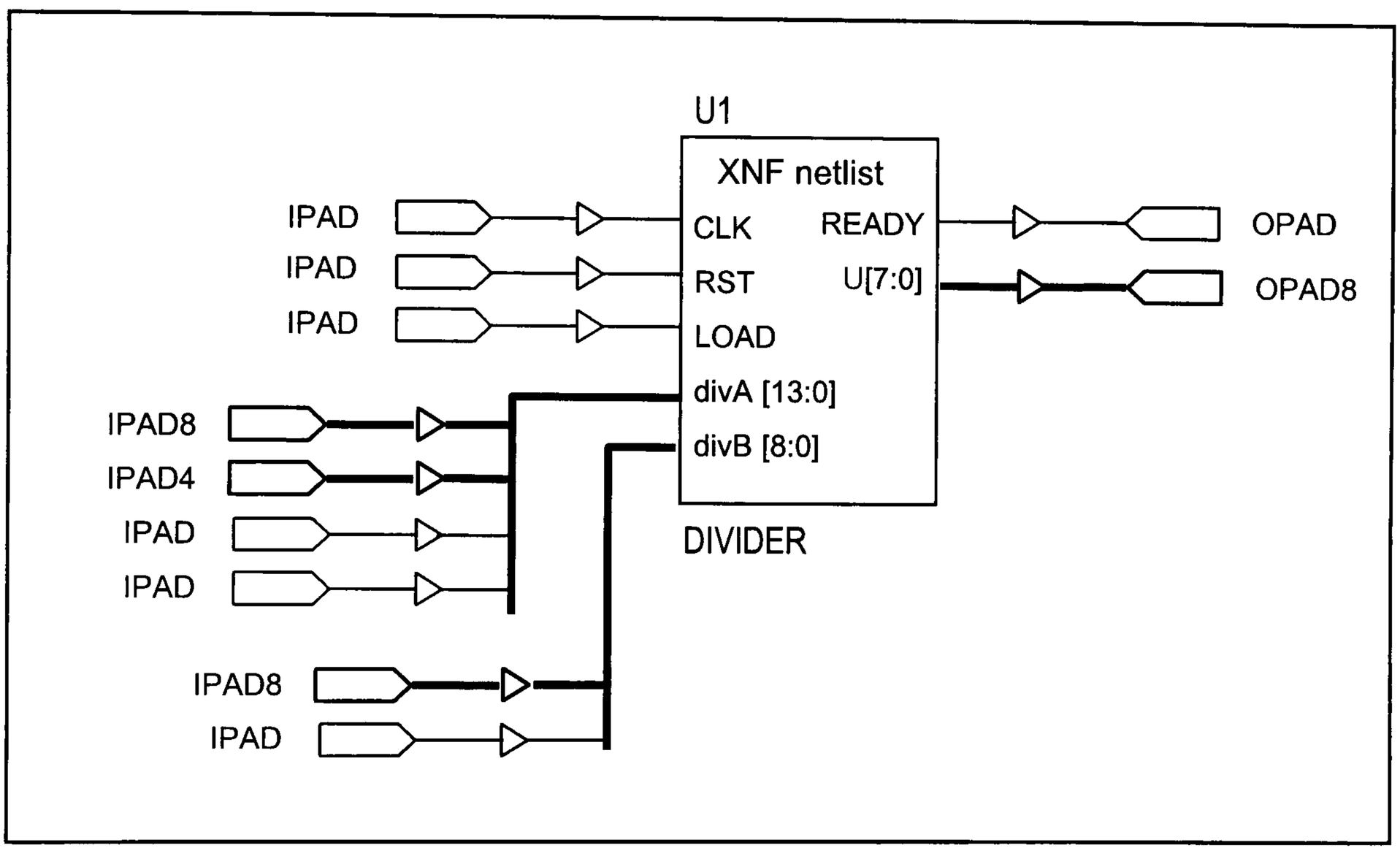


Figure 7-3 *Symbol of the component 'Divider' with I/O buffers and pads.*

7.3 Design Optimisation

Even after the behavioural design of the FLC is converted into a format which is fully supported for synthesis, there are still other issues to consider, particularly regarding implementation. This section looks at the question of *area optimisation*. *Figure 7-4* shows an extract from the implementation status report of the FLC design targeted at the Xilinx XC4010 FPGA.

```
>> Control_b.vhd (Behavioural)

Design Summary
_____

 Number of errors:              2
 Number of warnings:            3
 Number of CLBs:                  600 out of 400    150%
      CLB Flip Flops:
      CLB Latches:
      4 input LUTs:
      3 input LUTs:
 Number of bonded IOBs:          60  out of 63      95%
      IOB Flops:                0
      IOB Latches:              0
 Number of BUFGLSs:              1   out of 8       12%
 Number of RPM macros:          30
Total equivalent gate count for design:   6341
Additional JTAG gate count for IOBs:      2880
```

Not enough CLBs to support design

Maximum gate count is 10,000

Figure 7-4 *Synthesis Report for top hierarchy entity: Control.vhd*

The following points can be noted from the report:

- The FLC design requires more CLBs than is readily available in a Xilinx XC4010.
- The total equivalent gate count for the design is below the maximum count stated in the Data Book.
- The number of bonded IOBs in the device is sufficient for the design.

From these observations, it can be seen that although the design is too large for the Xilinx XC4010 in terms of CLBs, it has a total equivalent gate count that is within its limits. This suggests that the design is not fully utilising the logic gates in each CLB. The design can be easily implemented into one of the bigger devices in the Xilinx 4000 family, but in order to implement it into the XC4010XL, a certain number of modifications have to be made. These modifications optimise the design such that it is functionally identical to the original design but requires a smaller area. However, in reducing the area space, there is a trade off between other properties. In this case, one of the concerns is that by optimising the design, the fuzzy rule base also becomes more deeply integrated into its structure, thereby losing some of the features which make the controller generic and flexible.

7.3.1 Definition of Circuit Design Optimisation

The quality of a design can be measured by two main variables, *area* and *performance* [92]. The area of a design simply refers to the sum of the area space of the circuit components. The measure of performance is more complex as it involves analysis of the structure and behaviour of the circuit. There are a number of variables which relate to circuit performance such as propagation delay, cycle-time, latency and throughput but they are not discussed in great detail here. Briefly, *propagation delay* is the delay through the critical path of a circuit (for combinational logic circuit), *cycle-time* is the fastest clock period that can be applied to the circuit (for synchronous sequential circuit), *latency* is the number of clock cycles required to execute the operation and *throughput* refers to the rate at which data is consumed and produced by the circuit. *Optimisation* is the act of minimising the area and maximising the performance. Usually, there has to be a trade-off between the two. In most cases, design optimisation is subject to constraints such as:

- minimise the area under performance constraints
- maximise the performance under area constrains.

The task in this design falls under the first category, which is to minimise the area.

7.3.2 Structural Multiplication

The previous chapter shows how multiplication in VHDL programs can be carried out using the operator '*'. A simple multiplication by an arbitrary number, say five, can be performed with the following statement:

```
A<= B*5;
```

Another method of implementing this operation is to explicitly describe the step-by-step procedure of the binary multiplication.

Returning to the recent example, the pencil-and-paper method of the binary multiplication when B=9 is written as follows:

```
      1001    ← B
X      101    ← 5
-------------
      1001    ← B
     0000     ← 0
+   1001       ← B, shift left by 2 bits
-------------
    101101    ← A
```

This can be described in VHDL as:

```
A<= B + shl(B,"10");
```

where **shl(** B, "10" **)** is the command statement to shift the contents of **B** to the *left* by 2 bits.

The multiplication process effectively becomes a combination of an addition and a *shift-left* operation. Using this method to describe all the multiply-by-a-constant procedures, the area size of the design can be reduced significantly.

7.3.3 Optimising The Fuzzifier

In the fuzzification process of the FLC design that was presented in the previous chapter, the two inputs, x_1 and x_2, are processed using two separate fuzzification blocks. Since both of the inputs are fuzzified in exactly the same manner, it is possible to create just one fuzzification block to be shared between the two input variables. This reduces the area but it is at the expense of the performance, particularly *latency*. While previously, the entire process can be completed in just one clock cycle, this approach requires at least two clock cycles. To achieve this, there is also the need for memory elements to store the result of the first computation while the second set of data is being computed. At the end of the second computation, both sets of results are released simultaneously. In order to reduce the area size further, another method of optimisation is devised. The fuzzification block has 5 outputs, one for each fuzzy value defined in the inputs' universe of discourse. However, the fuzzification process entails that, for any single crisp value of the input x_i , only two adjacent fuzzy values are significant (with non-zero membership values). By ignoring the insignificant fuzzy values, the number of

output signals can also be reduced from five to two. The possible combinations of significant fuzzy values for an arbitrary input are:

$$B_i^1 \text{ and } B_i^2; \quad B_i^2 \text{ and } B_i^3; \quad B_i^3 \text{ and } B_i^4; \quad B_i^4 \text{ and } B_i^5.$$

It is found that using just three variables, ADRi, Bi_A and Bi_B, all the combinations can be sufficiently represented for any value of x_i as shown by the following statements:

$$\text{ADRi} = \text{“00”} : \quad \text{Bi\_A} = B_i^1 , \quad \text{Bi\_B} = B_i^2$$

$$\text{ADRi} = \text{“01”} : \quad \text{Bi\_A} = B_i^2 , \quad \text{Bi\_B} = B_i^3$$

$$\text{ADRi} = \text{“10”} : \quad \text{Bi\_A} = B_i^3 , \quad \text{Bi\_B} = B_i^4$$

$$\text{ADRi} = \text{“11”} : \quad \text{Bi\_A} = B_i^4 , \quad \text{Bi\_B} = B_i^5$$

Figure 7-5 illustrates how these conditions correspond with the universe of discourse.

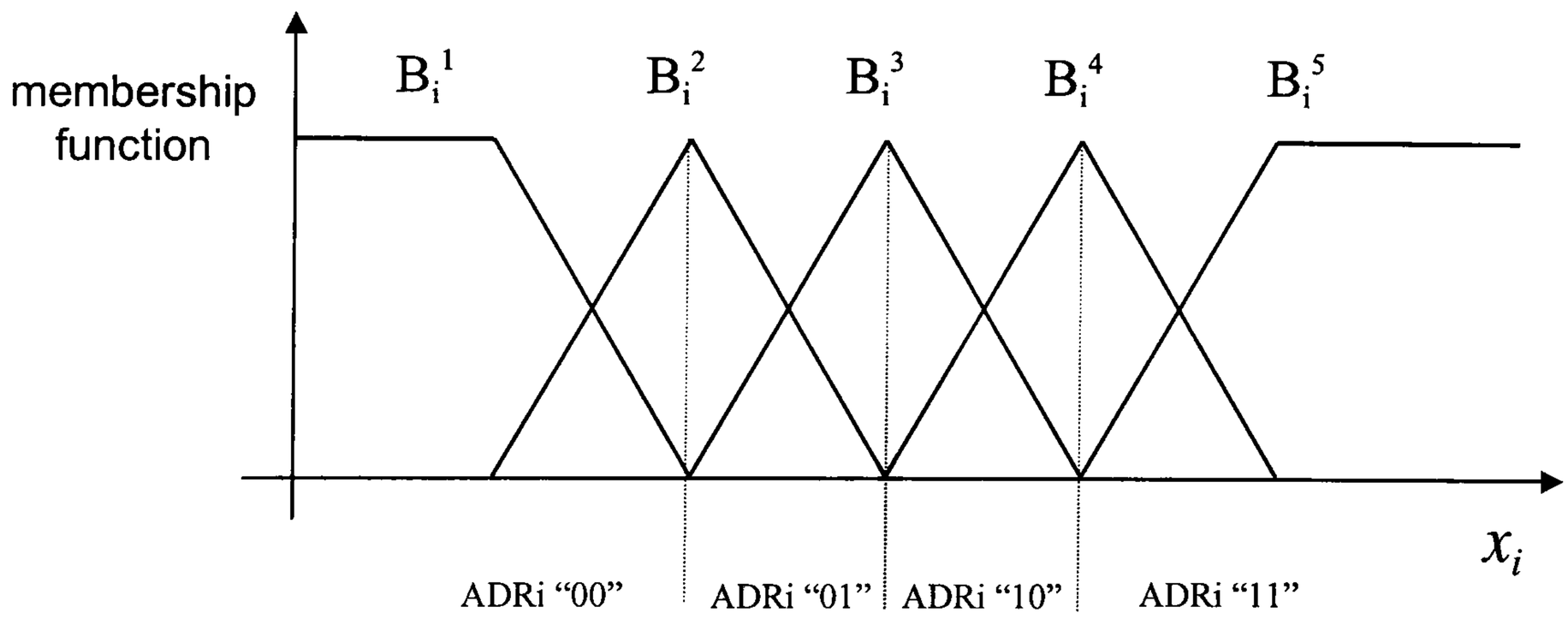


Figure 7-5 *Definition of Input Fuzzy Values.*

The complete VHDL code of the Fuzzifier can be found in *Appendix B-2*. An outline of the code's functional structure is described by the flow chart shown in *Figure 7-6*. Management of the input and output signals is mainly controlled by a status bit, **R_sig**. When **R_sig** is '1', the input **x1** is fuzzified but the output values remain unchanged. When **R_sig** is '0', the input **x2** is fuzzified and the new values of all the outputs are assigned. The variables **temp**, **temp_A** and **temp_B** represent the temporary registers used to store the fuzzy values of **x1** while the fuzzy values of **x2** are being computed. The operation takes two clock cycles to complete and all the fuzzy values are released simultaneously.

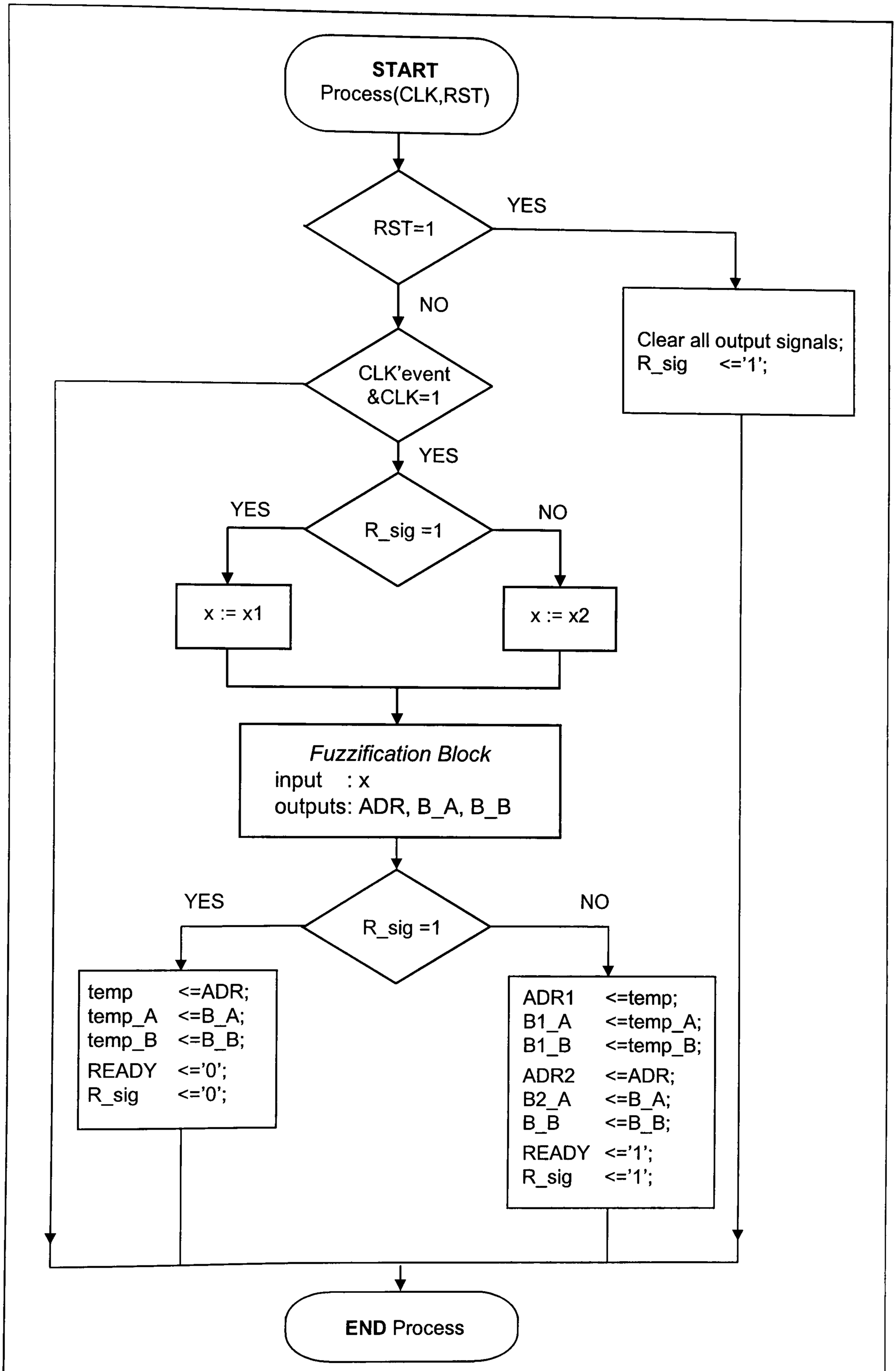


Figure 7-6 *Flow chart of the Fuzzifier.*

7.3.4 "Mini" FAM Tables

The FAM table of the FLC design discussed in *Chapter 6* is shown again here, in *Table 7-2*. It was mentioned that the inference of the fuzzy rules is achieved using Mamdani's inference technique and the VHDL code presented in that chapter uses an inference engine which triggers all 25 rules during every calculation. This section describes an algorithm which is developed to reduce the amount of computation required by focusing only on the relevant rules and ignoring those which are irrelevant to the conditions in question. From the previous section, it is known that for every set of inputs, only four fuzzy values (two for each input) are significant. This means that only four fuzzy rules are relevant at any one time.

Table 7-2 *FAM Table of the FLC Design.*

X_1 \ X_2	**NB**	**N**	**Z**	**P**	**PB**
NB	R^1 u = NVB	R^2 u = NB	R^3 u = N	R^4 u = NS	R^5 u = Z
N	R^6 u = NB	R^7 u = N	R^8 u = NS	R^9 u = Z	R^{10} u = PS
Z	R^{11} u = N	R^{12} u = NS	R^{13} u = Z	R^{14} u = PS	R^{15} u = P
P	R^{16} u = NS	R^{17} u = Z	R^{18} u = PS	R^{19} u = P	R^{20} u = PB
PB	R^{21} u = Z	R^{22} u = PS	R^{23} u = P	R^{24} u = PB	R^{25} u =PVB

NVB Negative Very Big
NB Negative Big
N Negative
NS Negative Small
Z Zero
PS Positive Small
P Positive
PB Positive Big
PVB Positive Very Big

An easier way of explaining the technique is to imagine the entire FAM table to be covered from view. Access to the content of the FAM table is only allowed through a small window and only four adjoining rules can be viewed through this window at a time. Therefore, instead of having to access 25 rules, the inference engine only has to access four rules during every computation. The window can move around the FAM table and its position is identified by an index j which is defined as follows:

ADR1="00" & ADR2="00" → $j = 0$

ADR1="00" & ADR2="01" → $j = 1$

ADR1=”00” & ADR2=”10” → $j = 2$

ADR1=”00” & ADR2=”11” → $j = 3$

ADR1=”01” & ADR2=”00” → $j = 4$

ADR1=”01” & ADR2=”01” → $j = 5$

…

…

ADR1=”11” & ADR2=”11” → $j = 15$

There are 16 ‘window positions’ altogether and the first six are shown in *Figure 7-7*. The shaded blocks are the rules which are considered relevant for the input conditions corresponding to the index j. To distinguish the ‘windowed’ view of the FAM table from the original table, the thesis shall refer to the smaller version as the *“Mini” Fuzzy Associative Memory (FAM) Table*.

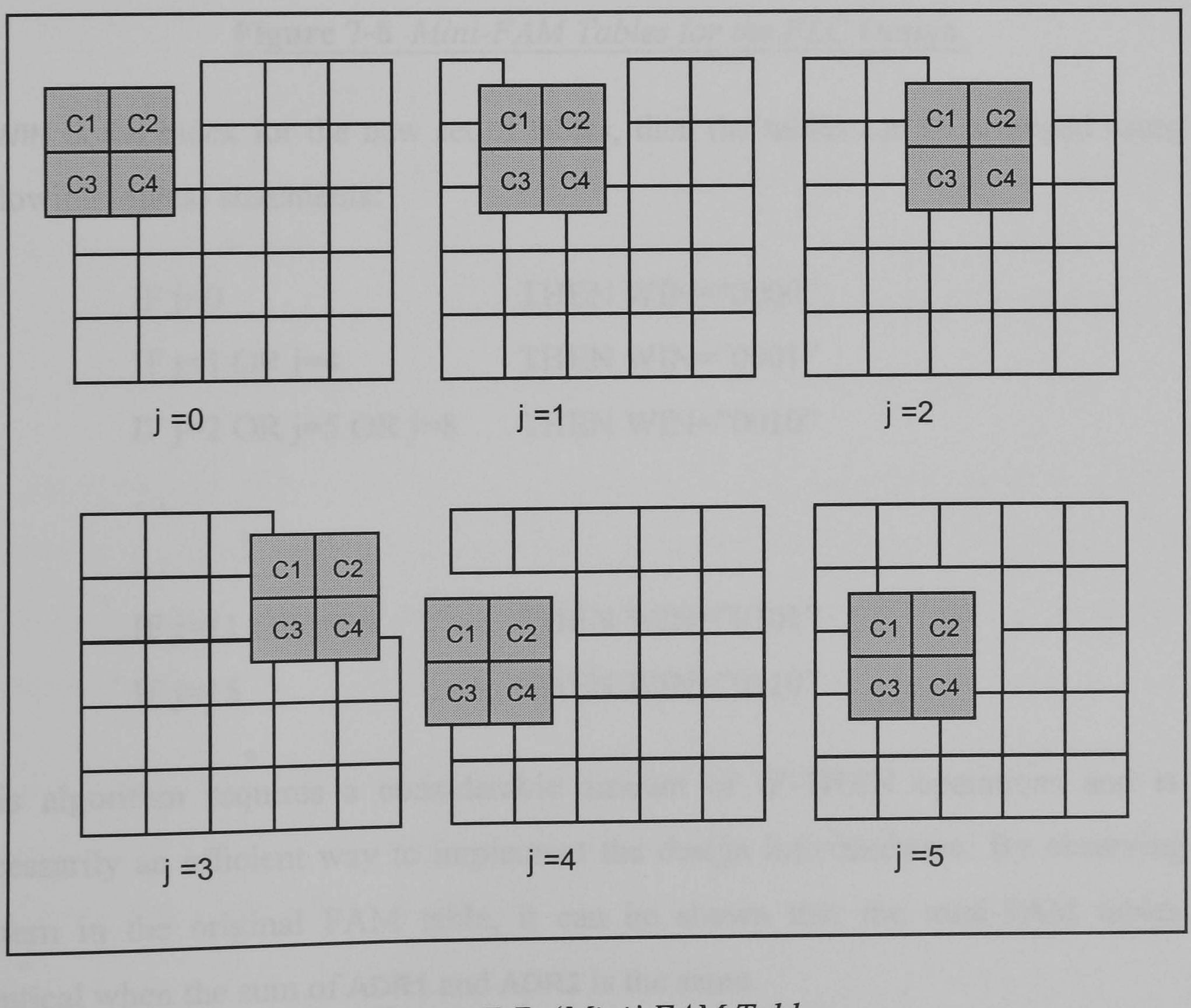


Figure 7-7 *‘Mini’ FAM Tables.*

When the window technique is applied to the FAM table in *Table 7-2*, it is observed that a number of the *mini-FAM* tables are identical (e.g. $j = 1$ and $j = 4$). Out of the 16 mini-FAM tables, there are only 7 unique tables as shown in *Figure 7-8*.

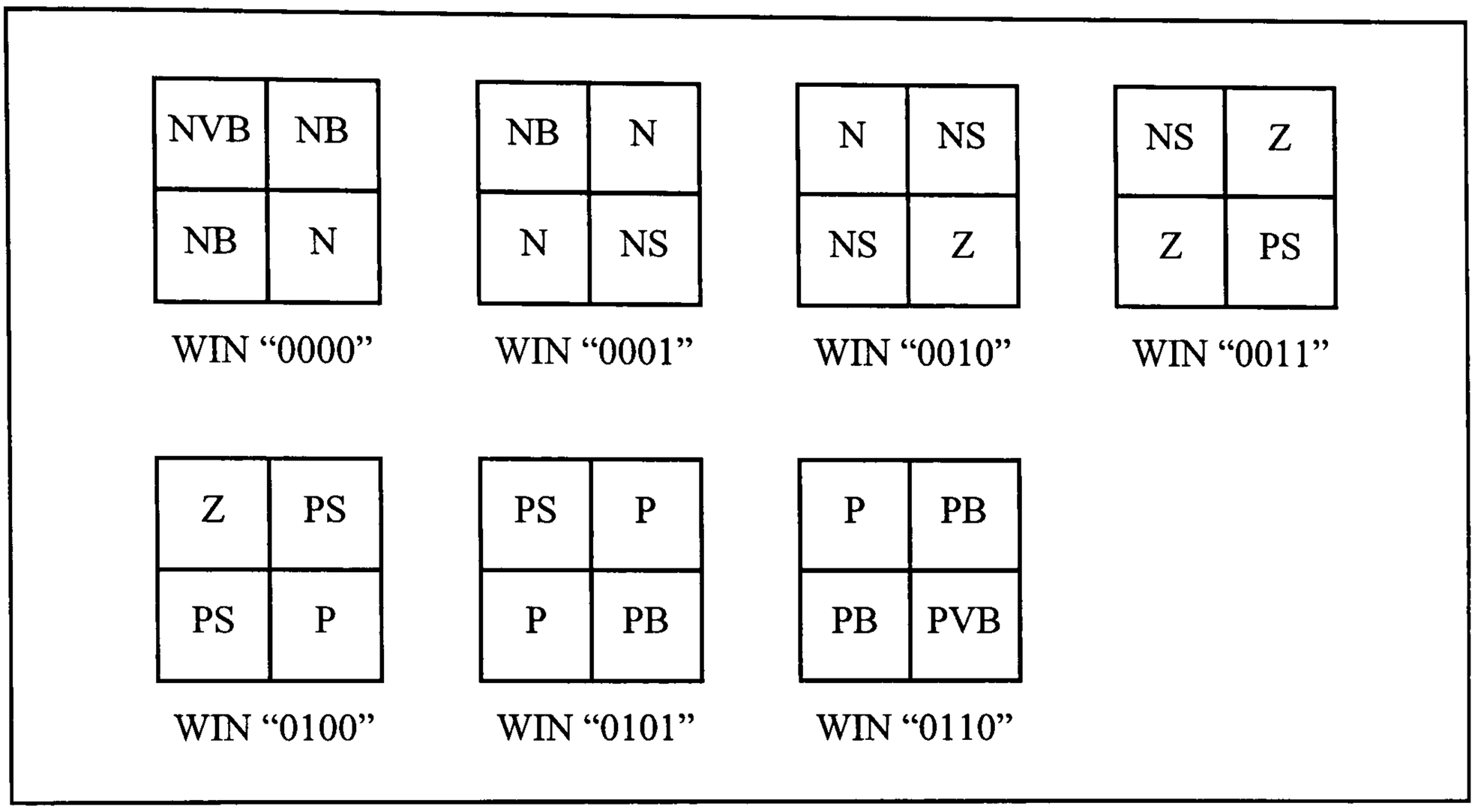


Figure 7-8 *Mini-FAM Tables for the FLC Design.*

If **WIN** is the index for the new set of tables, then the tables can be arranged using the following logical statements:

```
IF j=0                  THEN WIN="0000"
IF j=1 OR j=4           THEN WIN="0001"
IF j=2 OR j=5 OR j=8    THEN WIN="0010"
...
...
IF j=11 OR j=14         THEN WIN="0101"
IF j=15                 THEN WIN="0110"
```

This algorithm requires a considerable amount of IF-THEN operations and is not necessarily an efficient way to implement the design into hardware. By observing the pattern in the original FAM table, it can be shown that the mini-FAM tables are identical when the sum of **ADR1** and **ADR2** is the same.

Therefore, instead of using numerous IF-THEN operations, the arrangement of the mini-FAM tables is achieved using a single addition operation as shown by the following statement in the VHDL code (see *Appendix B-3)*:

```
WIN <= ("00"&ADR1) + ADR2;
```

where **ADR1** and **ADR2** are signals from the component **Fuzzify.** The function of the statement **("00"&ADR1)** is to expand the value of **ADR1** from 2 bits to 4 bits such that it is compatible with the 4-bit signal **WIN**.

The variables inside the mini-FAM table is subsequently processed in the section of the code that is marked '*Mini-Fuzzy Inference Engine*'. In the original code, the inference engine contains twenty five MIN-operations. The modified code consists of only four MIN-operations, which is a notable reduction.

7.3.5 Algorithm for Defuzzification

The original algorithm for the MAX-operation and defuzzification process contains two important computations:

- the aggregation of twenty five rule-consequents into 9 output (fuzzy) values
- the multiplication of each output value by a constant weighting.

By incorporating the modifications discussed in the previous sections, only four significant rule-consequents are considered. Therefore, the number of rule-consequents to be aggregated is reduced but the allocation of correct weightings for the significant output values becomes slightly more complicated.

From the tables in *Figure 7-8* it is obvious that regardless of the **WIN** value, the consequents C2 and C3 always point to the same fuzzy value (e.g. when **WIN**="0000": C1→NVB, C2→**NB**, C3→**NB**, C4→N).

This implies that only C2 and C3 have to be aggregated, hence:

DA= C1

DB= max[C2, C3]

DC= C4

where DA, DB and DC represent the membership function of the output fuzzy values.

The actual fuzzy values referred to by DA, DB and DC are determined by the value of **WIN**. If E^i is the weighting while VA, VB and VC are the weighted values, then

WIN="0000" : VA=DA* E^{NVB}, VB=DB* E^{NB}, VC=DC* E^{N} ;

WIN="0001" : VA=DA* E^{NB}, VB=DB* E^{N}, VC=DC* E^{NS} ;

WIN="0010" : VA=DA* E^{N}, VB=DB* E^{NS}, VC=DC* E^{Z} ;

WIN="0011" : VA=DA* E^{NS}, VB=DB* E^{Z}, VC=DC* E^{PS} ;

WIN="0100" : VA=DA* E^{Z}, VB=DB* E^{PS}, VC=DC* E^{P} ;

WIN="0101" : VA=DA* E^{PS}, VB=DB* E^{P}, VC=DC* E^{PB} ;

WIN="0110" : VA=DA* E^{P}, VB=DB* E^{PB}, VC=DC* E^{PVB} ;

Although the functions above can be implemented with seven IF-THEN statements it is preferable to adopt a less space consuming method. *Figure 7-9* shows a flow chart of the modified **Defuzz** design. The complete VHDL code can be found in *Appendix B*. In this design, instead of seven IF-THEN operations, only three are required to accomplish the main task (not counting the reset and clocking circuits).

The command statements in *Block 0* implement the reset conditions whereby all the outputs (except **READY**) and internal variables are cleared. The condition **LOAD**='1' and **READY**='1' indicates that the component is ready to initialise the defuzzification process and proceeds to execute the commands in *Block 1* and *Block 2*. *Block 1* performs the aggregation of C2 and C3 using a MAX-operator:

Block 1:

DA= C1

DB= max[C2, C3]

DC= C4

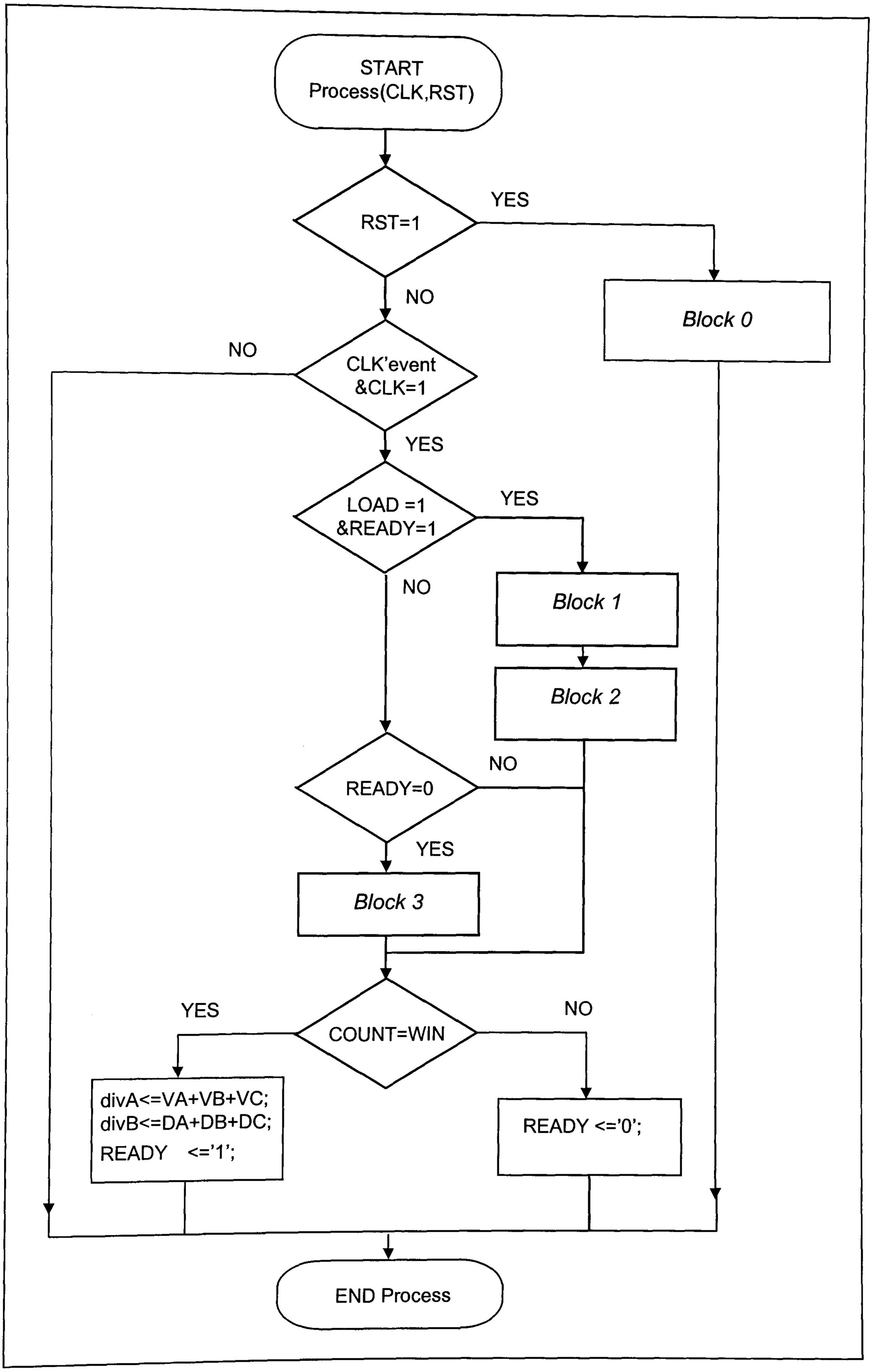


Figure 7-9 *Flowchart of the code for Defuzzification.*

In *Block 2,* it is assumed that **WIN**="0000", and the fuzzy values are then multiplied by the appropriate weightings for PVB, PB and P. If the weightings are defined as:

$$E^{NVB} = -40;\ E^{NB} = -30;\ E^{N} = -20;\ E^{NS} = -10;\ E^{Z} = 0;$$
$$E^{PS} = 10;\ E^{P} = 20;\ E^{PB} = 30;\ E^{PVB} = 40;$$

then *Block 2* can be written as:

Block 2:

VA = DA * -40

VB = DB * -30

VC = DC * -20

The subsequent commands check to see if **WIN** is in fact "0000". If the case is true (i.e. **COUNT**=**WIN**), then the computation is complete and the output signals **divA**, **divB** and **READY** are assigned with the appropriate values. Otherwise, the computation is incomplete (**READY**='0') and *Block 3* is executed in the next clock cycle.

The strategy of this algorithm relies on the fact that the values of VA, VB and VC increases steadily as **WIN** increases. In other words,

IF

WIN= "0000": VA^0=DA*(-40), VB^0=DB*(-30), VC^0=DC*(-20)

THEN,

WIN="0001": $VA^1=VA^0$ +(DA*10), $VB^1=VB^0$ +(DB*10), $VC^1=VC^0$ +(DC*10)

WIN="0010": $VA^2=VA^1$ +(DA*10), $VB^2=VB^1$ +(DB*10), $VC^2=VC^1$ +(DC*10)

WIN="0011": $VA^3=VA^2$ +(DA*10), $VB^3=VB^2$ +(DB*10), $VC^3=VC^2$ +(DC*10)

....

....

By taking advantage of the recurring pattern, *Block 3* can be described with just four logical statements:

Block 3:

COUNT=COUNT+1

VA=VA+(DA*10)

VB=VB+(DB*10)

VC=VC+(DC*10)

Once the optimised structural-level design is successfully completed, it is implemented using Xilinx Foundation HDL Editor.

7.4 Implementation

Each element of the FLC is designed and carefully optimised for synthesis. Five VHDL components **DERIV**, **FUZZIFY**, **INFER**, **DEFUZZ** and **DIVIDER** make up the core of the fuzzy controller. The nature of the components' connection and the functionality of the processes are described by an upper hierarchy VHDL code **Control.vhd**. The diagram in *Figure 7-10* illustrates how these components are wired to each other to form the complete control system. In addition to the components, two synchronous processes represented by the blocks **Process1** and **Process2** are used to synchronise various signals. The code is subsequently synthesised to generate a netlist of the upper hierarchy component **Control**.

In creating a single top hierarchy component, it is easier to proceed into the implementation stage using the Xilinx Foundation Schematic Editor whereby the design can be developed in a graphical form. The '***Create Macro symbol from netlist***' function allows the component **Control** to be converted into a Macro symbol that can exist in the Schematic Editor. Then, using Xilinx Foundation Implementation tools, the design can be compiled into a bitstream file.

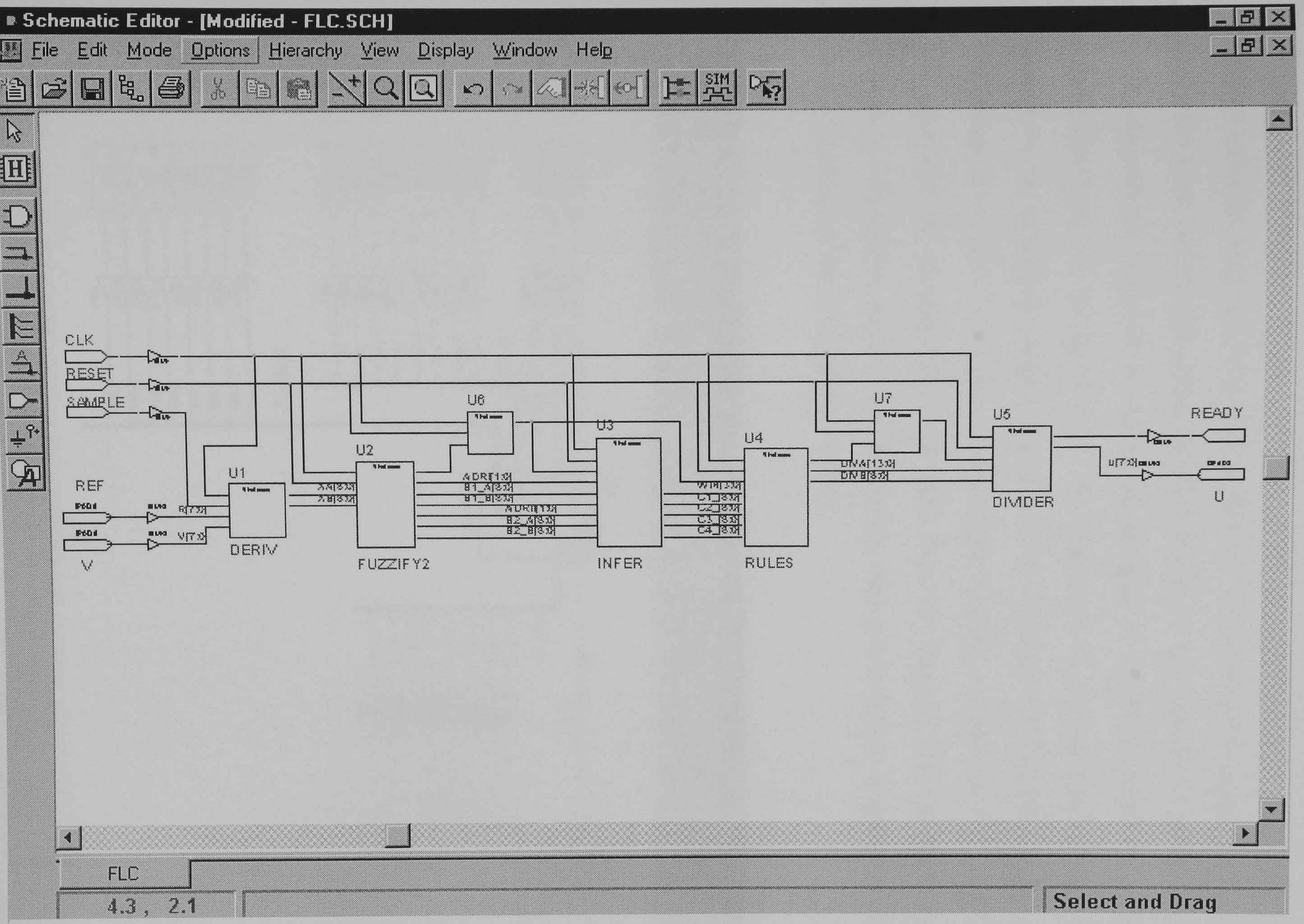


Figure 7-10 *Illustration of the construction of* **Control.**

Xilinx XC4010 FPGA is available in several packages and the one used for this design is the PC-84 package which has 84 I/O pins in total. During the generation of the bitstream, the inputs and outputs of the design are mapped to the physical I/O pins of the FPGA. The allocation of pin numbers can either be performed automatically by the implementation tools or explicitly specified by the user. In this design, all the pins are specified by manually assigning the appropriate pin numbers to the IPADs and OPADs in the Schematic Editor. This enables the designer to have full control over the function of the physical pins in the FPGA. The allocation of pin numbers to the I/O pads is shown by the schematic diagram (as seen in the Xilinx Foundation Schematic Editor) of the design in *Figure 7-11*. The numbers (preceded by the letter 'p') in the I/O pads are the allocated pin numbers. It can be seen from the diagram that the clock input of the design is allocated to pin 35, and as it is the main clock signal, a global buffer IBUFG is used instead of the normal input buffer.

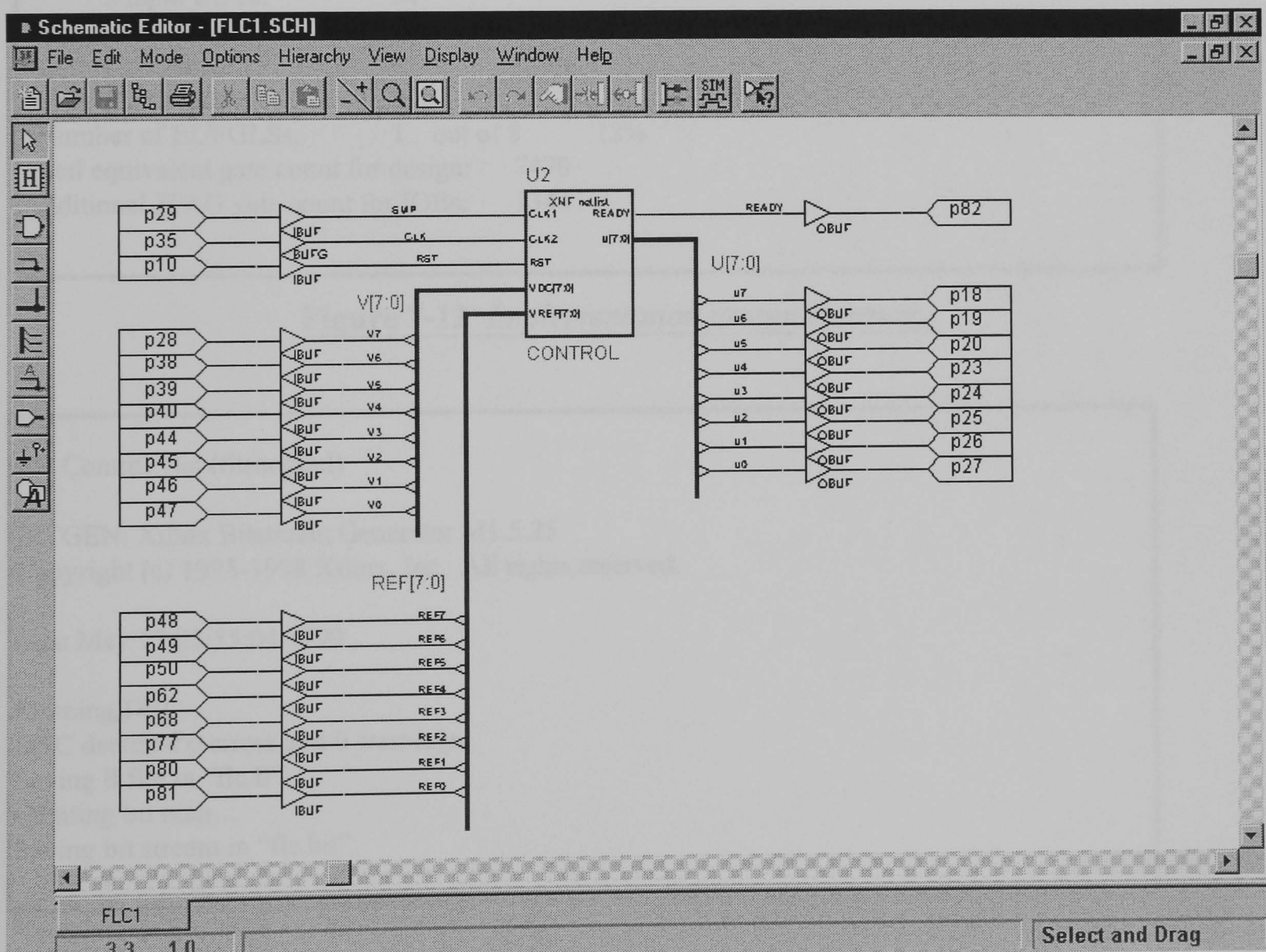


Figure 7-11 *Xilinx Foundation Schematic Editor design.*

Once the hardware specifications have been confirmed, the netlist is compiled into a bitstream file using the *Implementation* procedure in Xilinx Foundation Project Manager. *Figure 7-12* shows the status report of the implementation process. By comparing the results with the report in *Figure 7-4*, it is obvious that the modifications were effective in utilising the logic blocks in the FPGA in a more efficient manner. The report shown in *Figure 7-13* confirms that the bitstream file is successfully generated without errors and is ready for downloading.

```
>> Control.vhd (Structural)

Design Summary
--------------

   Number of CLBs:                 386 out of 400  96%
      CLB Flip Flops:              317
      CLB Latches:                 0
      4 input LUTs:                677
      3 input LUTs:                94
   Number of bonded IOBs:          28  out of 63   43%
      IOB Flops:                   0
      IOB Latches:                 0
   Number of BUFGLSs:              1   out of 8    12%
Total equivalent gate count for design:     7470
Additional JTAG gate count for IOBs:        1344
```

Figure 7-12 *Implementation design summary.*

```
>> Control.vhd (Structural)

BITGEN: Xilinx Bitstream Generator M1.5.25
Copyright (c) 1995-1998 Xilinx, Inc.  All rights reserved.

Mon May 17 23:55:04 1999

Running DRC
DRC detected 0 errors and 0 warnings.
Saving ll file in "flc.ll".
Creating bit map...
Saving bit stream in "flc.bit".
```

Figure 7-13 *Bitstream generation report.*

The circuit board used to house the target FPGA during downloading is the XS40 Board, a Xilinx FPGA evaluation board from *X Engineering Software Systems (XESS) Corps*. *Figure 7-14* shows a picture of the XS40. Further information can be found in [93]. The XS40 Board connects to the parallel port of a Personal Computer (PC) via a cable with DB-25 connectors. Having set up the board (including the configuration jumpers) appropriately in accordance to [93], the bitstream file can then be downloaded into the FPGA using the XSTOOLS software.

The downloading of the FLC design in the Xilinx FPGA represents a major milestone in the hardware realisation of the control system. Subsequent work, described in the next chapter, focuses on the assembly of the various hardware components of the system as well as the practical tests conducted on the complete system using the FPGA controller.

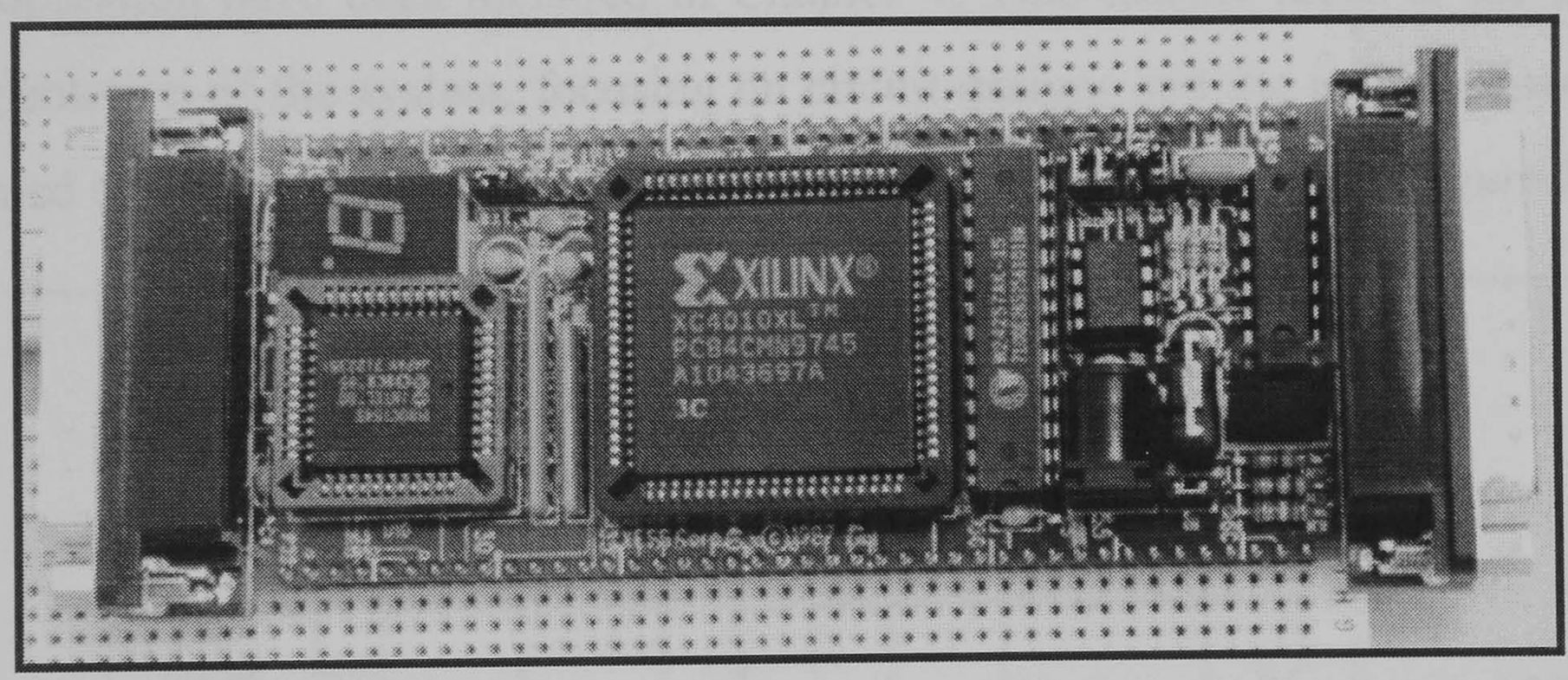


Figure 7-14 *Picture of XS40Board.*

Chapter
8

System Assembly and Practical Tests

An introduction to the complete generator-control system and its principle of operation have been included in *Chapter 4*. This chapter looks at the hardware implementation of the system, focusing on circuit designs and the results of some tests performed on the controller. A block diagram of the system is shown in *Figure 8-1*.

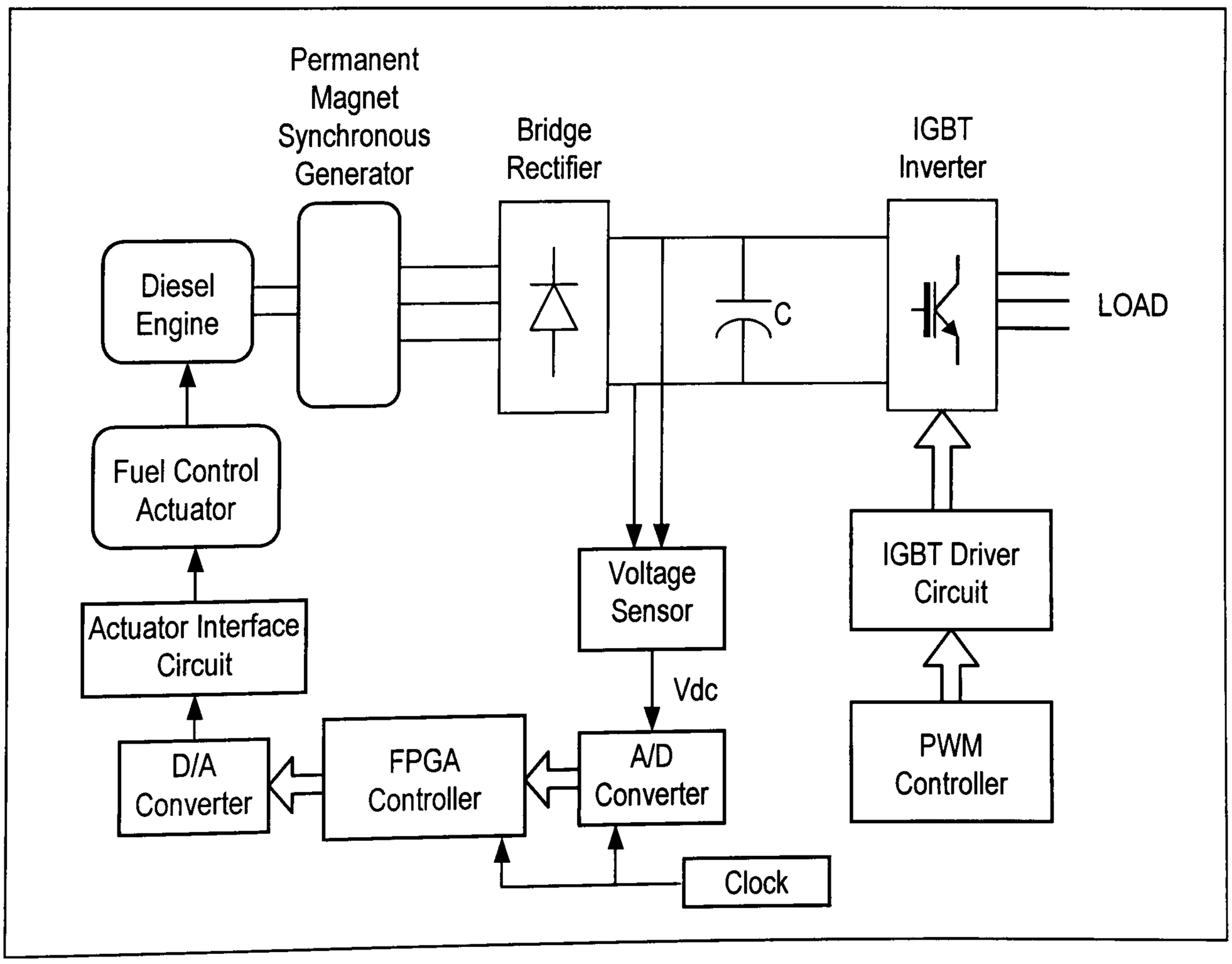


Figure 8-1 *Block diagram of engine-generator set and controller.*

The system can be subdivided into 3 main sections:

- an *electromechanical system* which comprises the generator, the diesel engine and the electromagnetic actuator
- a *power electronic system* consisting of the diode bridge rectifier, d.c. link and IGBT inverter
- an *electronic control system*.

The *electronic control system* can be further divided into two components, namely the PWM controller and a control loop consisting of the sensing and interfacing circuits as well as the FPGA (fuzzy) controller.

The engine-generator set used to test the controller is supplied by *Newage International* and some of the tests presented were carried out at the *Department of Engineering, University of Durham*.

8.1 The Electromechanical System

The generator used for the experiments is a permanent magnet synchronous machine with three phase star connected windings. The phase resistance and inductance of the machine are 0.6Ω and 1.0mH respectively. The generator is coupled to a 3-cylinder diesel engine which provides the driving power. The rate of flow of diesel fuel into the engine is controlled by an electromagnetic actuator which positions the fuel rack. The position of the fuel rack, and hence the rate of fuel flow, is a function of the amount of current flowing into the actuator coil. Since it is easier to control voltage than current, the most obvious solution is to supply a controllable d.c. voltage to the coil. This is feasible because the d.c. impedance of the coil is constant since only the resistive effect is significant. Therefore, from Ohm's Law, the actuator current I_{ACT} is directly proportional to the voltage across the actuator coil V_{ACT}.

This approach is employed in a simple experiment to test the characteristics of the engine-generator set and also to determine the magnitude of the actuator current required for the operation of the system. Initial tests reveal that the maximum position of the fuel rack is achieved with an actuator current of 5A. *Figure 8-2* shows a block diagram of the layout for this experiment.

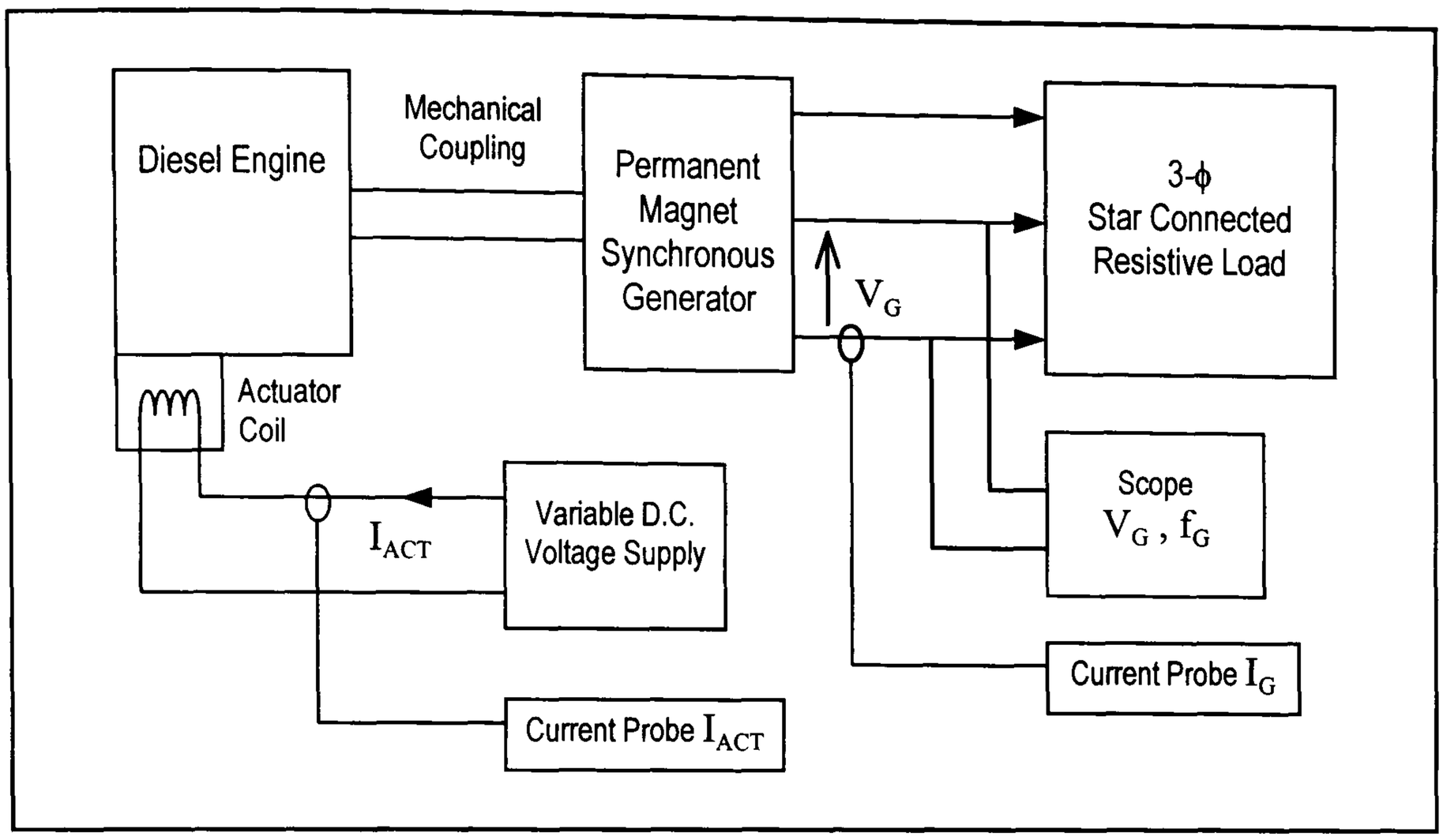


Figure 8-2 *Block diagram of the electromechanical system test.*

Using different load values, the actuator current I_{ACT} is manually controlled such that the output voltage and frequency remain within a reasonable range. The result is shown in *Table 8-1*. It can be observed that the actuator current required for the operating conditions tested is in the order of 2.0A. *Figure 8-3* shows the waveform of the generator's terminal voltage V_G.

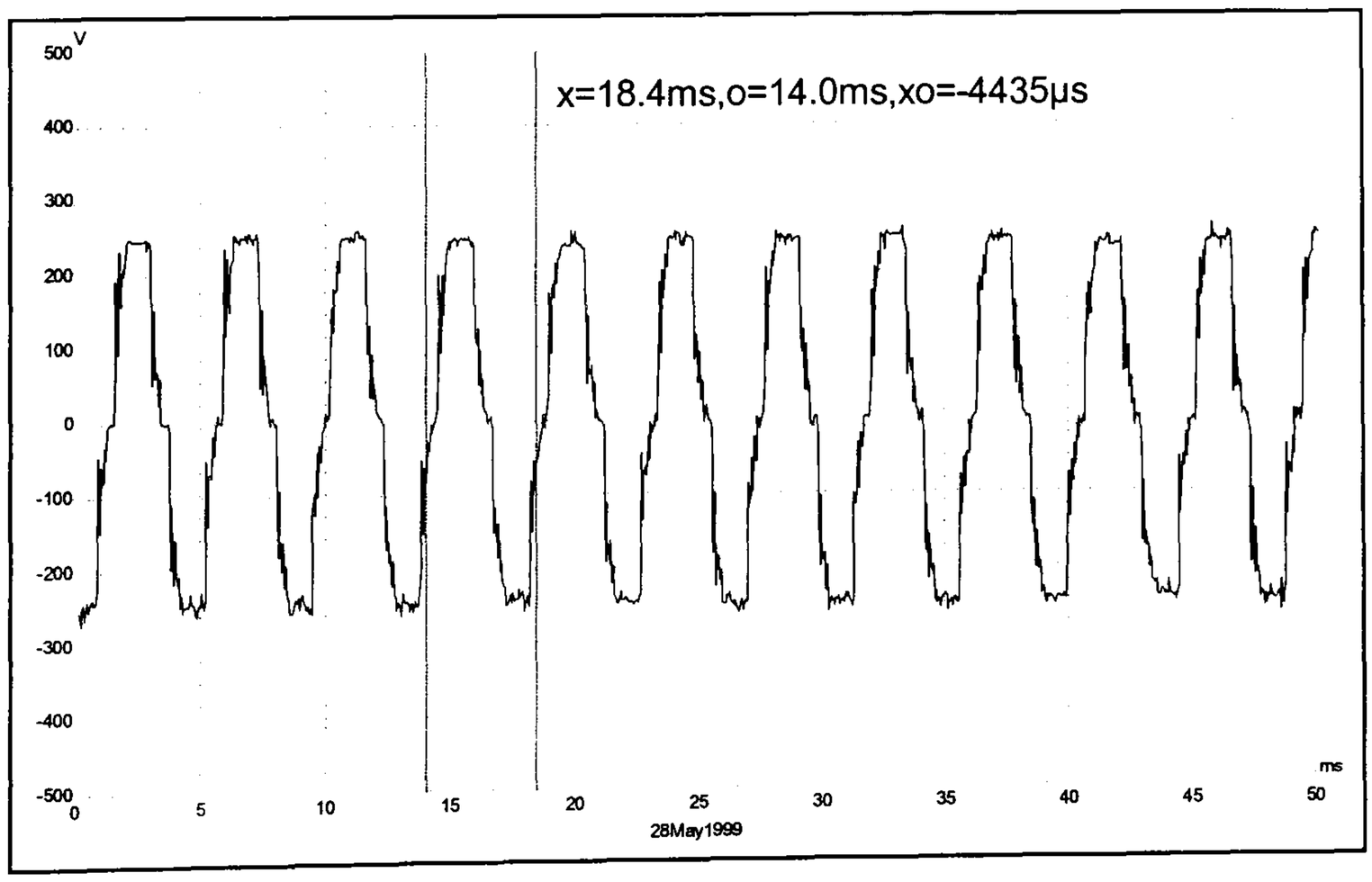


Figure 8-3 *Voltage waveform at generator terminal.*

In this experiment, it is possible to use a variable d.c. voltage supply and manually adjust the actuator current. However, in a feedback control system, a slightly different strategy is adopted. The reason for this is the difficulty in supplying a continuously variable voltage output capable of sourcing currents of the order of a few amperes. Variable voltage outputs in electronic circuits usually have limited current capabilities (of the order of hundreds of milli-amperes) while larger current sources often have fixed voltages. Therefore, in feedback control system of this nature, a Pulse Width Modulation (PWM) technique is adopted. It is based on the same principle as the sinusoidal-PWM scheme used for inverter control. This is discussed in greater detail in *Section 8.3.5*.

Table 8-1 *Result of electromechanical system test.*

Load: 81.5 Ω per phase

I_C [A]	1.8	1.9	2.0	2.1
$I_{G\,(PEAK)}$ [A]	2.2	2.5	2.6	2.7
$V_{G\,(PEAK)}$ [V]	260	290	305	310
frequency, f_G [Hz]	270.0	312.5	333.3	357.1

Load: 56.0 Ω per phase

I_C [A]	1.9	2.0	2.1	2.2
$I_{G\,(PEAK)}$ [A]	2.5	3.0	3.2	3.5
$V_{G\,(PEAK)}$ [V]	240	280	300	320
frequency, f_G [Hz]	263.2	312.5	333.3	384.6

Load: 47.0 Ω per phase

I_C [A]	1.9	2.0	2.1	2.2
$I_{G\,(PEAK)}$ [A]	x	3.4	3.6	4.0
$V_{G\,(PEAK)}$ [V]	x	260	280	320
frequency, f_G [Hz]	x	303.0	312.5	357.1

Load: 40.2 Ω per phase

I_C [A]	1.9	2.0	2.1	2.2
$I_{L\,(PEAK)}$ [A]	x	3.6	4.2	4.4
$V_{G\,(PEAK)}$ [V]	x	260	280	300
frequency, f_G [Hz]	x	294.1	312.5	333.3

Load: 35.2 Ω per phase

I_C [A]	1.9	2.0	2.1	2.2
$I_{G\,(PEAK)}$ [A]	x	x	4.6	4.8
$V_{G\,(PEAK)}$ [V]	x	x	270	280
frequency, f_G [Hz]	x	x	312.5	322.6

x - the given control current cannot sustain operation of the engine-genset under the loading condition causing the engine to stall.

8.2 Power Electronic System

Power converters play an important role in electric drive systems and power supply networks. They are used mainly to convert electrical energy from one form to another. In this project, two types of power conversion are used in the power electronic system:

- a.c. to d.c. conversion
- d.c. to a.c. conversion.

Various options of power components are available for this application, ranging from a *total discrete solution* to a *total integrated solution*. The former involves building the converters almost from scratch, using discrete components such as thyristors, power diodes, Mosfets and transistors. The advantages of this option are the design flexibility and the ability to change a single damaged component without the need to replace the entire converter. The disadvantages include an increased amount of time and effort in design as well as a relatively large footprint. An alternative, the *total integrated solution* approach, takes advantage of the latest power electronic innovations whereby system designs are integrated into *application specific power electronic modules*. An example relevant to the project is *Semikron*'s MiniSkiip. This device is an entire d.c. link converter system (rectifier and inverter) moulded into a single module. It is an elegant design approach which reduces the overall footprint as well as the effort and time involved in design. However, the repair cost can be extremely high as the entire module would have to be replaced if just one 'component' in the module is damaged. The

design in the present project is a prototype which may be subjected to stressful tests or future modifications, therefore the *total integrated solution* is not an appropriate choice. Instead, a 'middle' approach is employed, that is, the power electronic system comprises two integrated modules.

A rectifier module is used to perform the a.c. to d.c. power conversion. The device selected for this task is the 36MT80, a 35A three phase uncontrolled rectifier from *International Rectifier*.

D.c. to a.c. conversion is achieved using a three phase bridge power inverter. Power inverters are utilised in a wide variety of power electronic applications, especially in the control of a.c. motor drives and uninterruptible power supply systems. In this project, an inverter is used at the output end of a d.c. link to maintain the desired a.c. voltage and frequency. The *Semikron* 'SKM40GD123D' three phase inverter module is selected for this purpose. It is made up of six IGBT switches with built-in free wheeling diodes and they are arranged in a three phase bridge configuration as shown in *Figure 8-4*. Each IGBT has a maximum collector-emitter voltage rating of 1200 volts and a continuous collector current rating of 40 Amperes (at case temperature of 25°C). IGBTs are selected for this project because they offer a combination of the characteristics of bipolar transistors and Mosfets, having low conduction losses and high switching frequency ratings.

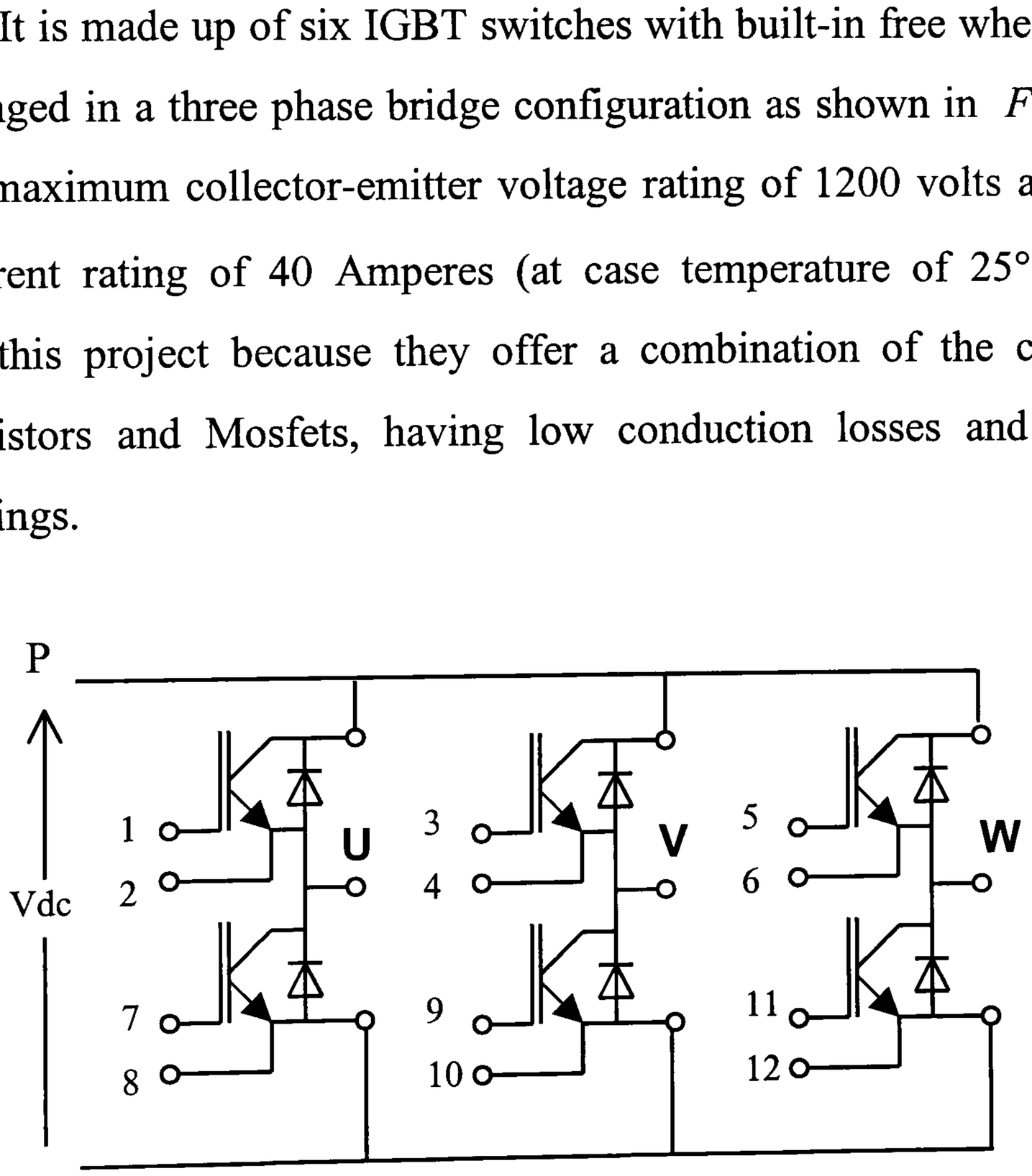


Figure 8-4 *Semikron IGBT Inverter module.*

IGBT Driver Circuit

A driver circuit is required as the interfacing stage between the control signals and the inverter. It serves two main purposes:

i. Isolation.

The six signals from the control board share one common ground but because the driving signal to each IGBT gate has to be isolated from one another, the control signals cannot be directly connected to the IGBTs. The driver board isolates the individual signals. However, the three lower IGBTs in the bridge are shorted at their emitter (pins 8, 10 and 12 in *Figure 8-4)*, therefore the signals to these transistors can share a common ground.

ii. Amplification.

The outputs of the control board are of TTL level type (at 5V). This is not enough to drive the IGBTs. Therefore, the second function of the driver board is to step up the driving voltage. The maximum threshold voltage of the SKM40GD123D is 6.5 volts. In this design, the driver circuit produces switching signals at 15V, which is appropriately above the threshold voltage.

Figure 8-5 shows a circuit diagram of the driver board. The circuit can be divided into four main branches. Transformers **T1** and **T2** provide isolated power to the four circuit branches so that the reference level (ground) for the outputs can be independent of one another. The **REG** ICs are voltage regulators which maintain steady 15v d.c. supplies to the circuit. The optoisolators - marked **OP** in the diagram - provide complete isolation of the input signals from driver circuit. Finally, Motorola IGBT drivers MC33153 (**D** in the circuit diagram) are used to supply the driving voltage and current.

In *Figure 8-5*, the output pins are marked with a number that corresponds to its connection in the inverter shown in *Figure 8-4*. The last three pairs of outputs (pins 7 to 12) can share the same power supply (and hence the same reference level) because the emitter of the IGBTs which they are connected to are tied together. Driving three IGBTs with the same supply may seem to be placing more stress on the rectifier and voltage regulator than those of the other branches. However, it is still within the capability of the devices and is not shown to affect the circuit performance.

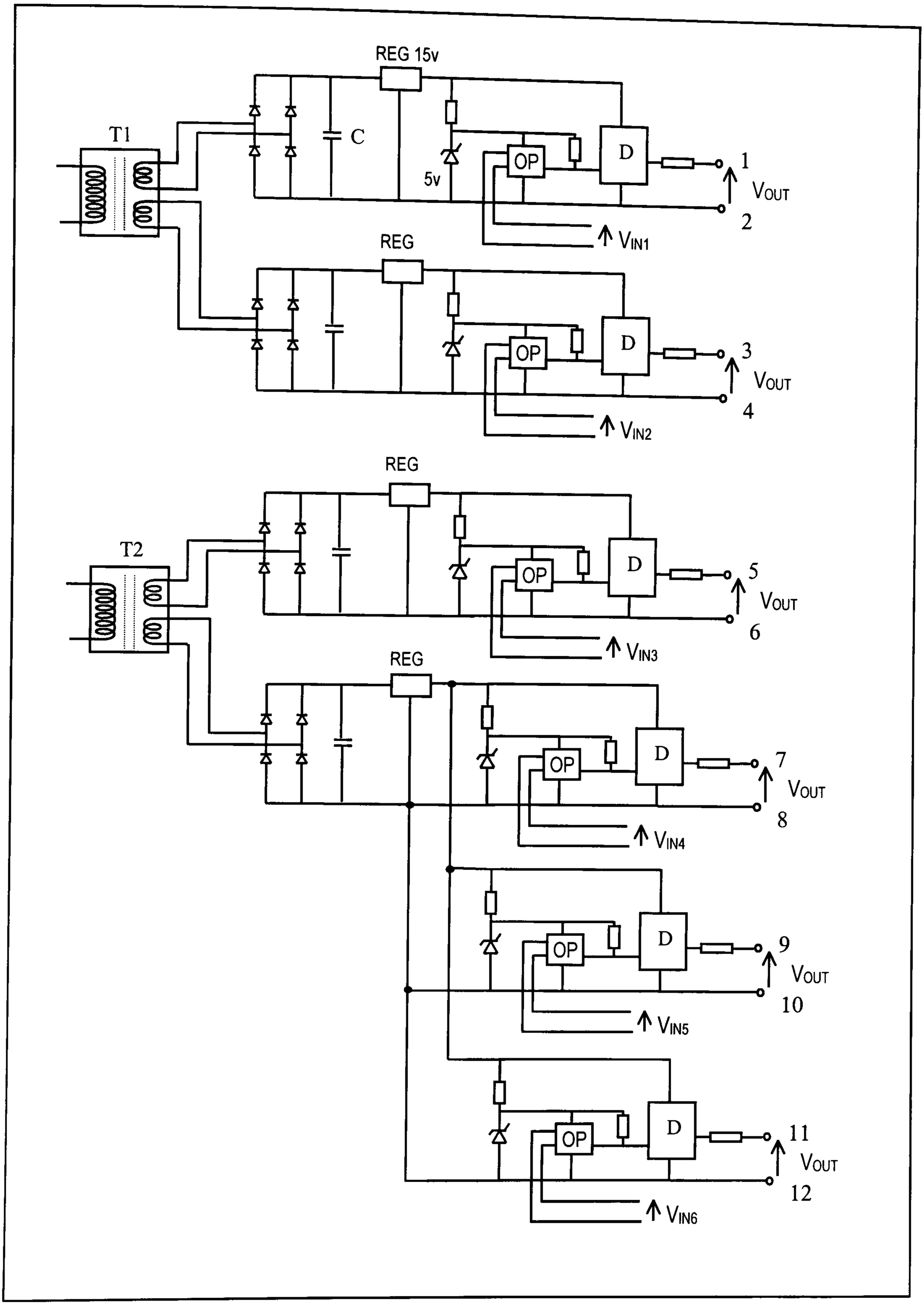


Figure 8-5 *IGBT driver circuit.*

8.3 Sensing and Interfacing Circuits

This section discusses the circuits which are used to complement the control systems which were presented earlier in the thesis.

8.3.1 Voltage Sensor

While it is necessary for the control circuit to sense the voltage at the d.c. link, care has to be taken such that the circuit is properly protected from the relatively high voltage of the power section. For this purpose, a Hall effect voltage transducer is used. The device is essentially a voltage transducer which utilises the Hall effect principle, whereby current is measured by taking advantage of the magnetic field generated by a current carrying conductor.

The voltage sensor used in this project is the LV25-P, a PCB-mounting Hall-effect voltage transducer. A diagram of the transducer is shown in *Figure 8-6*. The LV25-P provides the necessary galvanic isolation between the power circuit of the d.c. link and the corresponding low voltage signal to the control system. It has a nominal input current $I1_N$ of 10mA and a turns ratio of 2500:1000.

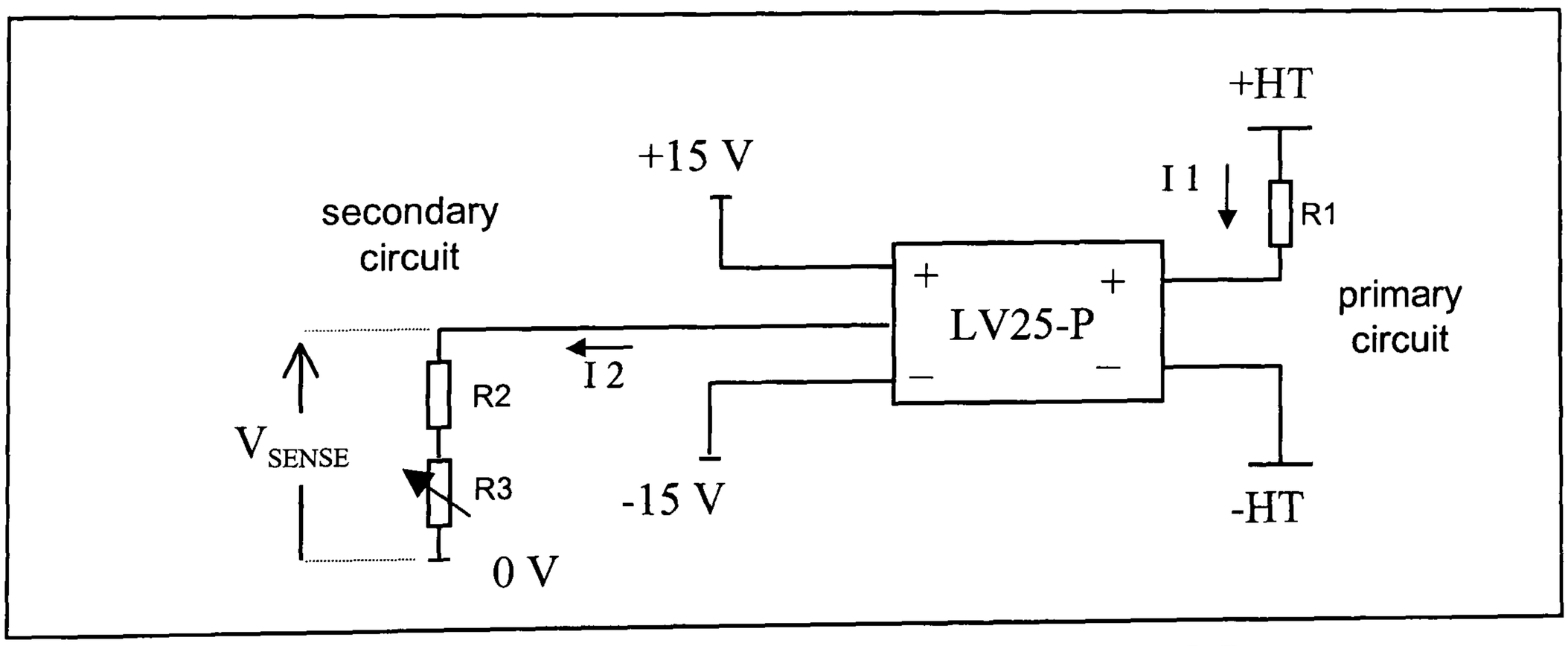


Figure 8-6 *Hall effect voltage transducer, LV25-P.*

The primary resistor, R1 is calculated such that the nominal voltage to be measured corresponds to a primary current of 10mA. From the turns ratio, this produces a nominal output current of 25mA. The nominal power voltage is assumed to be 250 volts d.c.

Therefore,

$$R1 = \frac{250}{10x10^{-3}} = 25k\Omega$$

Allowing for a 20% increase in voltage,

$$I1_{MAX} = 1.2 \times 10mA = 12mA$$

$$P = I1_{MAX}^{2} \times R1 = 3.6\ W$$

Therefore the chosen resistor R1 should have a power rating above 3.6W.

The values for R2 and $R3_{MAX}$ are chosen to be 150kΩ and 100kΩ respectively. By tuning the value of R3, a suitable power voltage to output signal ratio can be calibrated. The sensor circuit is built and calibrated for a voltage ratio of 100:1.

8.3.2 Analogue to Digital Conversion

The conversion of the analogue signal from the transducer into digital signal is achieved using National Semiconductor's ADC0804 IC. The device is an 8-bit CMOS A/D converter based on the *successive approximation* conversion technique. The logic inputs meet TTL specifications as well as CMOS. It also has an on-chip clock generator which eliminates the necessity for an external clock and the conversion time is 100μs. *Figure 8-7* shows a simplified schematic diagram of the conversion circuit utilising the ADC0804.

The signal from the transducer is connected to the Vin(+) pin of the A/D converter while the digital output is fed into the FPGA via a buffer. The signals connected to the $\overline{RD}$ and $\overline{WR}$ pins come from a clock and pulse generation circuit (refer to *Section 8.3.4*).

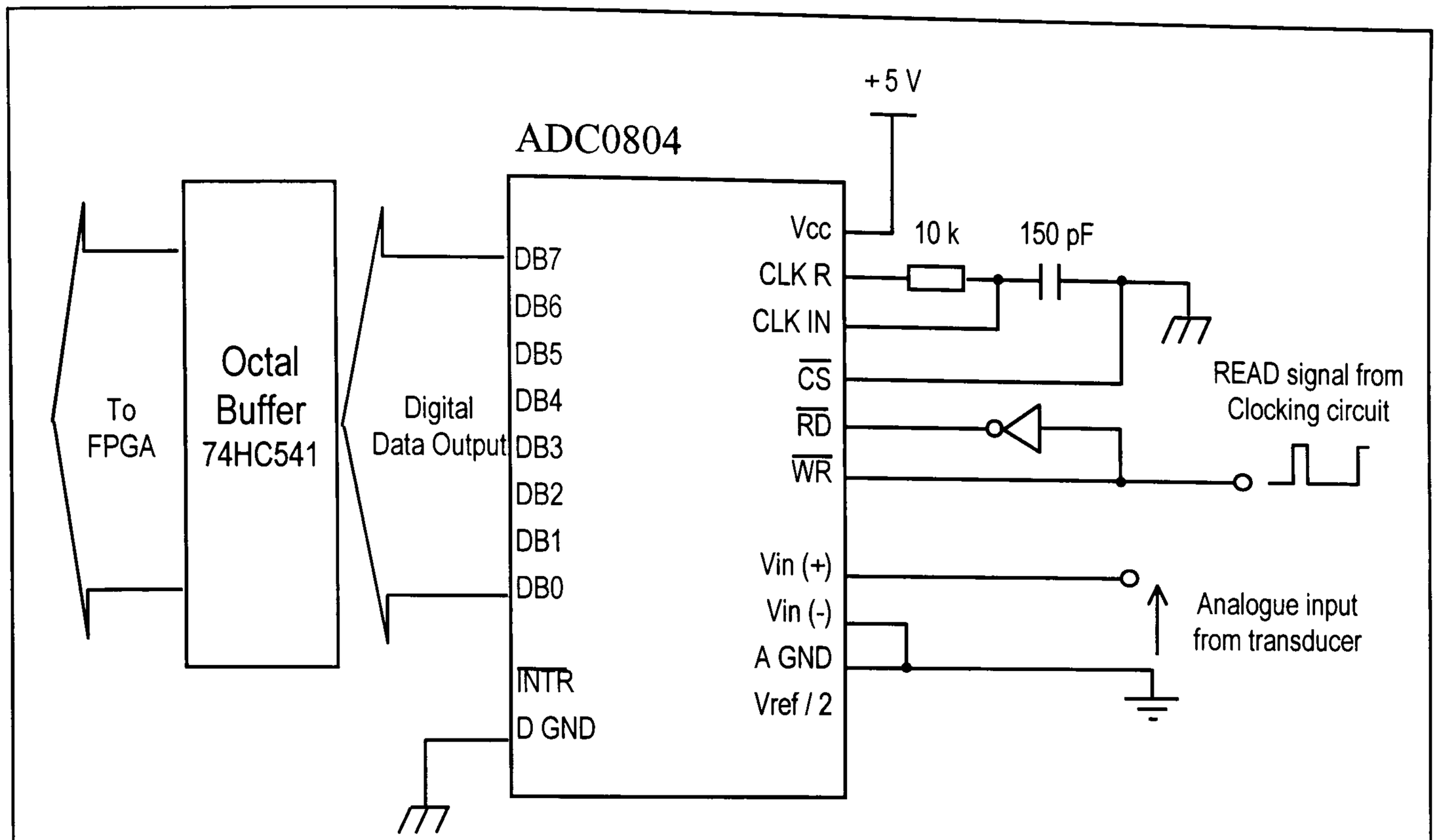


Figure 8-7 *Analogue to digital conversion circuit.*

8.3.3 Digital to Analogue Conversion

The ZN425E is used to convert the digital output of the FPGA controller into an analogue signal. It is essentially an 8 bit digital to analogue converter i.c. based on a R-2R ladder network. A schematic of the converter circuit is shown in *Figure 8-8*. A 741 operational amplifier is used to amplify the output signal of the ZN425.

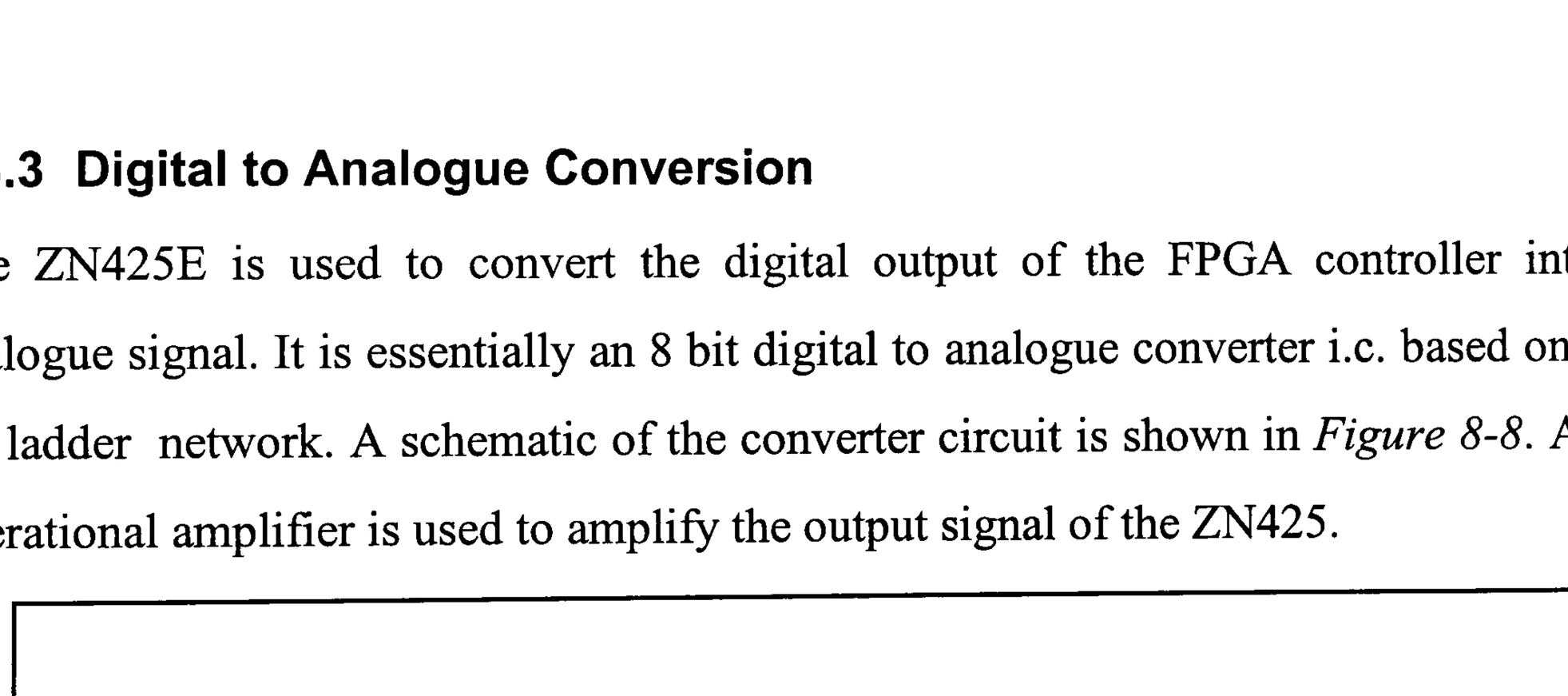


Figure 8-8 *Digital to analogue converter circuit.*

8.3.4 Clocking Circuit

Like all synchronous circuits, the FPGA design described in *Chapter 7* requires a clocking signal. This is provided by an oscillating crystal (X) as shown by the circuit in *Figure 8-9*. This circuit can be divided into two sections, an *oscillator circuit* and a *pulse generation circuit*. The former is a standard CMOS clock signal generation circuit which uses a crystal to provide the oscillation while the latter generates the appropriate clock and pulse sequence for the FPGA and A/D converter. It is entirely possible for the pulse generation circuit to be incorporated into the FPGA design, but in this project, it is designed as a separate circuit. The signal from the oscillator circuit is used to drive a CMOS4040 counter via the point A in the circuit. A waveform diagram of the pulse generation circuit outputs (CLK, READ and SMP) is shown in *Figure 8-10*. The signal CLK is used as the primary clocking signal for the FPGA. Taken from the Q0 output of the counter, CLK is a reflection of the oscillator circuit frequency.

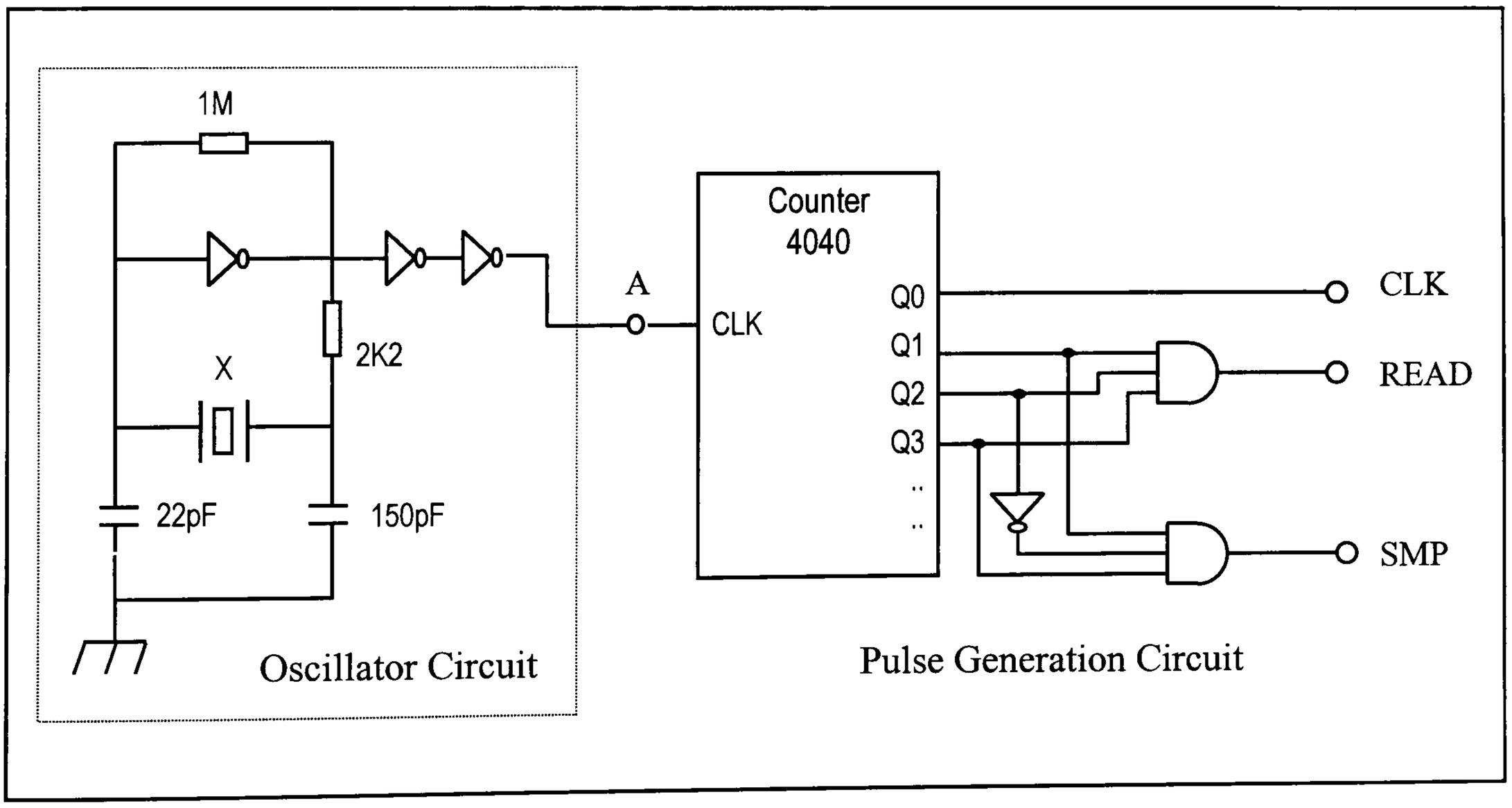


Figure 8-9 *Clocking circuit.*

In the meanwhile, the READ pulse initiates the A/D conversion process in the ADC0804 while the SMP pulse enables the FPGA to sample the converted digital signal from the A/D converter. The sequence of these two pulses is such that the FPGA takes a sample of the digital value at the output of the converter just after conversion. Based on the the conversion time of the ADC0804, it is necessary for the time lapse between the

READ pulse and the SMP pulse to be more than 100μs for conversion to be complete. This corresponds to a maximum clock frequency of 10kHz.

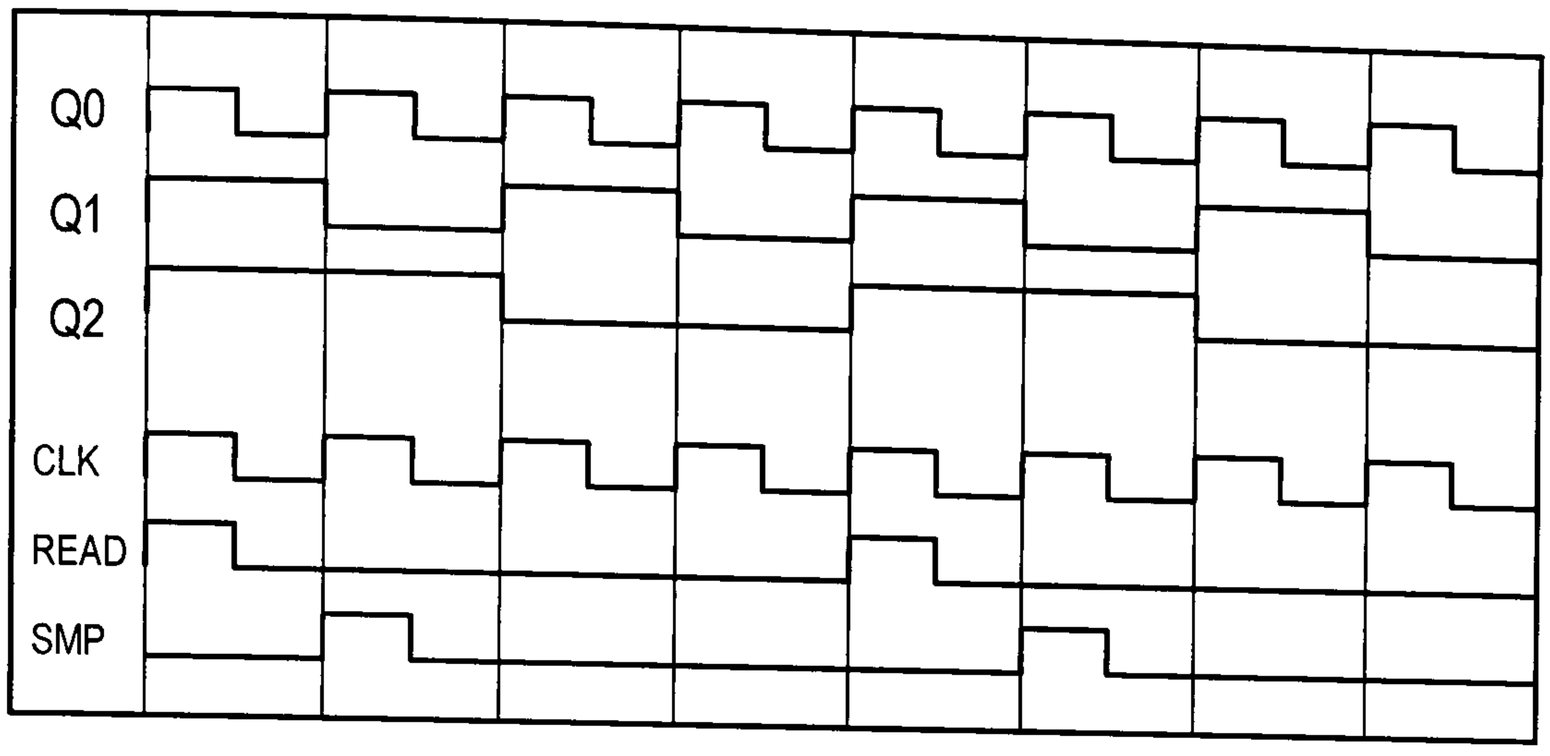


Figure 8-10 *Waveforms of Pulse Generation Circuit.*

8.3.5 Actuator Interface Circuit

In the experiment described in *Section 8.2*, the actuator is controlled manually, using a variable d.c. voltage source. This approach is not practical in feedback control systems for reasons already explained. The Pulse Width Modulation (PWM) technique is used instead to control the actuator. It is similar to the sinusoidal-PWM inverter control technique described in *Chapter 5*, except that, instead of a sinusoidal power waveform, the (analogue) control signal is used as the reference.

Figure 8-11 shows how the PWM signal can be obtained by comparing a saw-tooth carrier waveform with a d.c. reference voltage. The ON time of the PWM signal t_{ON} is proportional to the d.c. reference voltage (within a range), while the switching frequency is fixed by the carrier frequency. The circuit shown in *Figure 8-12* implements this technique and is used to interface the controller output with the fuel control actuator. A waveform generator i.c., the ICL8083, provides a triangular carrier waveform which is adjustable in terms of frequency and slope. The analogue output from the D/A converter is amplified in two stages then used as the reference signal of the PWM circuit. The gain can be controlled from the variable resistor R2 while R6 allows the offset voltage to be adjusted.

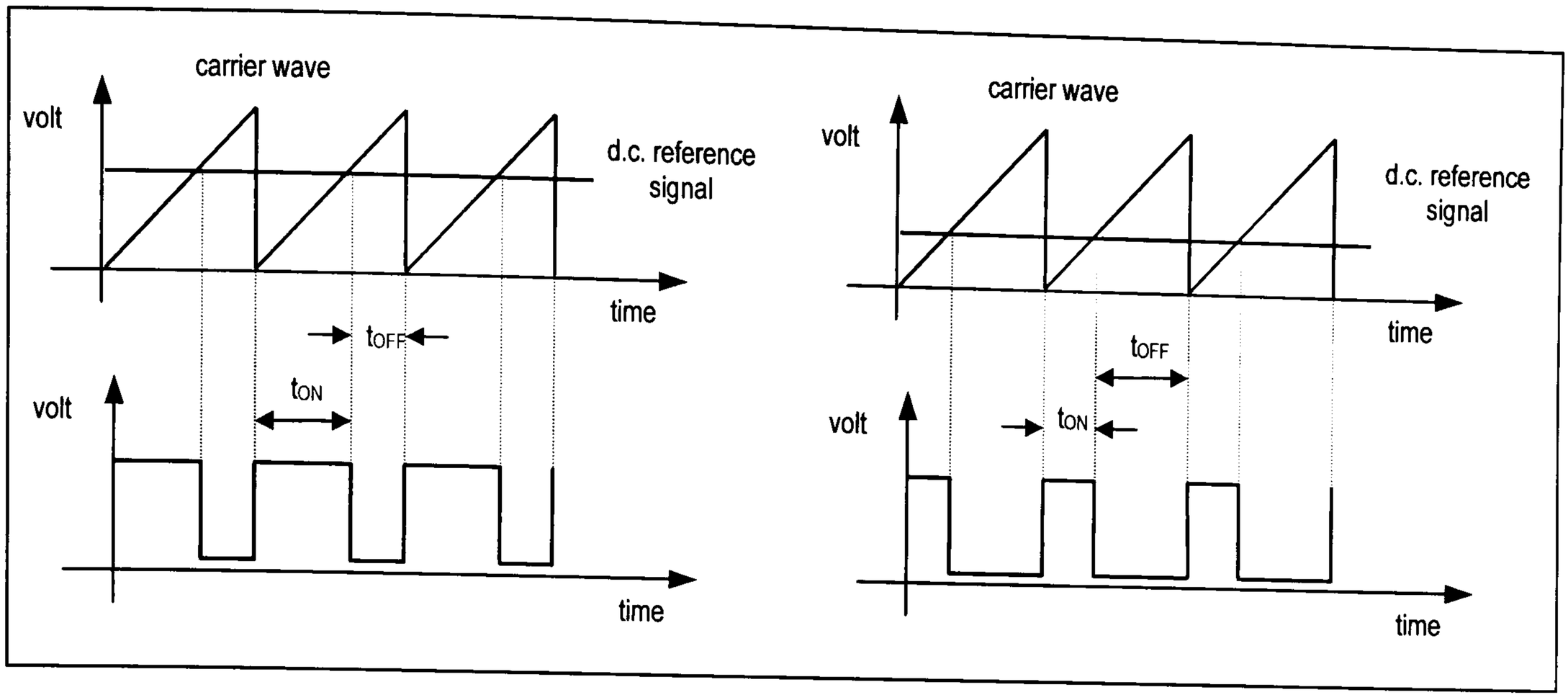


Figure 8-11 *Generation of PWM switching patterns.*

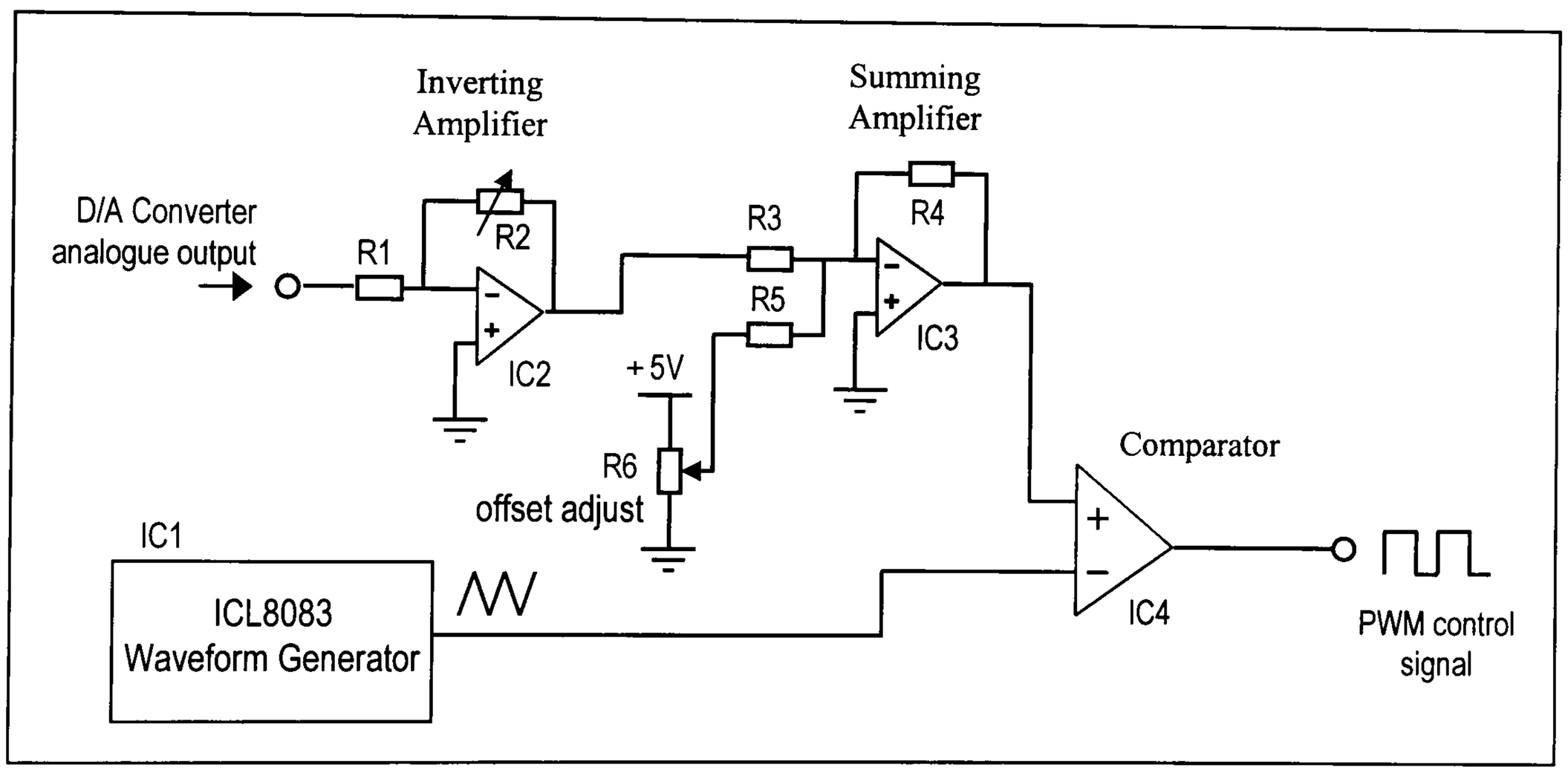


Figure 8-12 *PWM waveform generation circuit.*

The PWM output of this circuit is then used to control the actuator via the circuit shown in *Figure 8-13*. Here, the power to the actuator is provided by a d.c. power supply. By switching the MOSFET on and off appropriately, the actuator current can be governed. However, due to the inductive property of the actuator coil, any attempt to switch the MOSFET off and sever the flow of current would be resisted by the coil, and may cause disruption to the operation if the circuit is not properly designed. Placing a diode D, parallel to the coil opens a channel for the current to flow when the power supply is cut

off. The result is two different current waveforms in the circuit: I_1 is a square wave similar to the PWM signal while I_{ACT} is a continuous current waveform as a result of the smoothing effect of the coil.

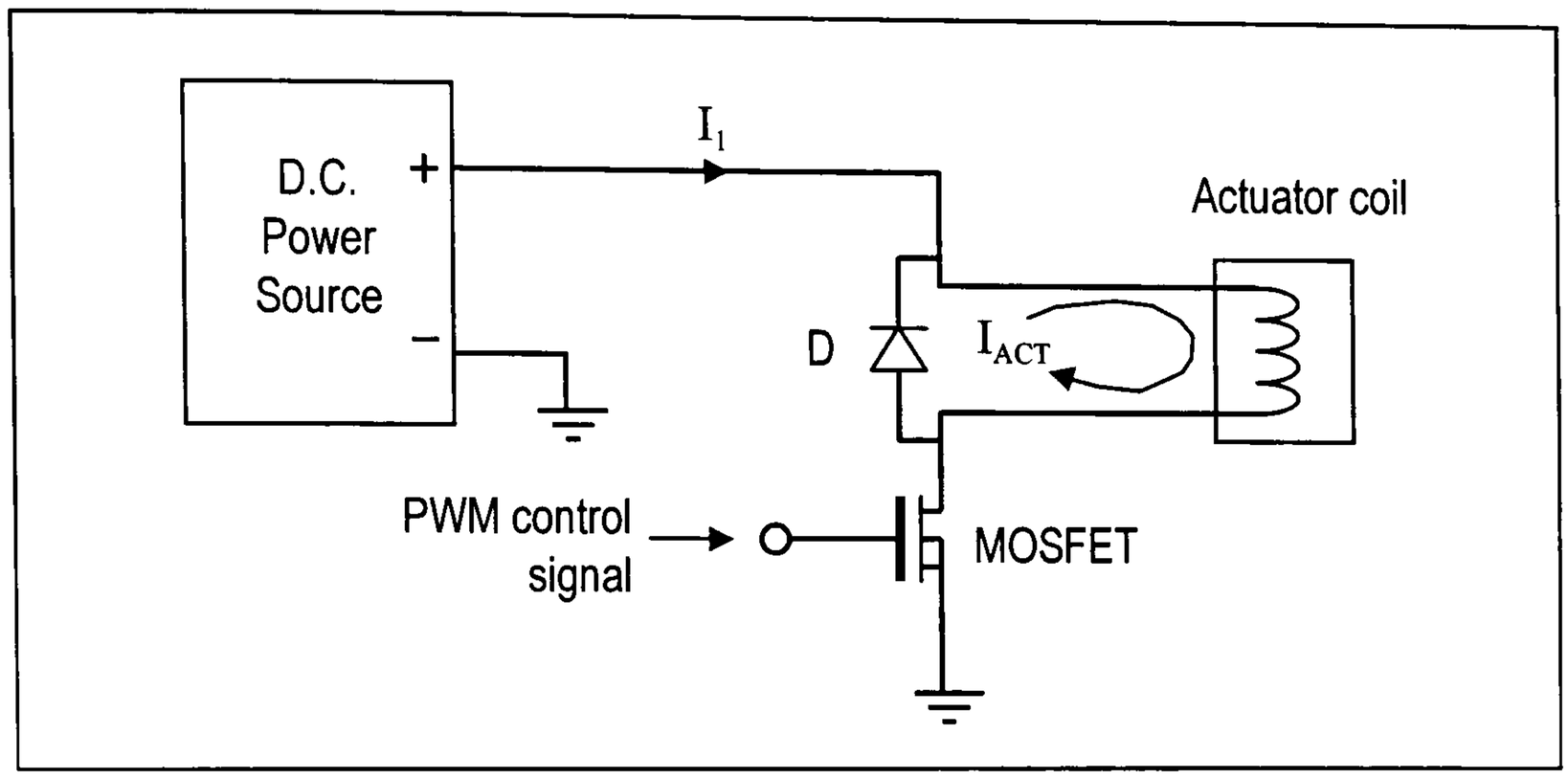


Figure 8-13 *Circuit for actuator control.*

8.3.6 Power Supply

To complete the system, a commercial power supply unit is used to provide the necessary power to all the circuits in the system. The unit selected for this purpose is the LPT45 module from ASTEC/RS Components. It has three output voltages of +5V, +15V and -15V with maximum loads of 4A, 2A and 0.5A respectively. The unit is found to suit the requirements of all the circuit boards in the system.

8.4 Hardware Tests

8.4.1 Selecting clock frequency

The control law of the fuzzy controller can be loosely represented by *Eq. 8-1*, an expression derived from PI-control. Although the explanation in the following paragraphs will move along this line of argument, it is important to be aware that this equation is only a model of the fuzzy controller and does not represent the actual internal mechanism of the controller itself. The control actions in the fuzzy controller are derived, not from this equation, but from a set of fuzzy rules.

Eq. 8-1 $$\Delta u = \mathrm{K_P} \cdot \Delta e + \mathrm{K_I} \cdot e$$

From *Eq. 8-1*, the control signal is given by the sum of its previous value and the output of the fuzzy controller as shown by *Eq. 8-2*.

Eq. 8-2 $$u = uz^{-1} + \Delta u$$

The resulting effect is a control signal which increases or decreases in gradual steps between samples until the desired output is achieved. Since the sampling period (nT) is a function of clock frequency, the rate of change of the control signal $\frac{\mathrm{d}u}{\mathrm{dt}}$, - not to be confused with Δu which is the change in u in one sampling period (nT) - is also a function of the clock frequency. In other words, while Δu remains independent of the clock frequency, the value of $\frac{\mathrm{d}u}{\mathrm{dt}}$ is at least partially dependent on it. This concept can be pictorially described by the diagram in *Figure 8-14*, where the two output signals have the same Δu but different rate of change due to different clock frequencies (1/T).

This characteristic allows the response of the controller to be modified simply by adjusting the clock/sampling frequency. In some ways, parallels can be drawn between the effect of selecting a clock/sampling frequency and the effect of tuning the gains in a PI-controller, although they are essentially two different operations. A high clock frequency, equivalent to using a large proportional gain, is expected to increase the

response time but induce oscillation into the system. Naturally, a low clock frequency would have the opposite effect. The aim is to obtain a balance between the two extremes.

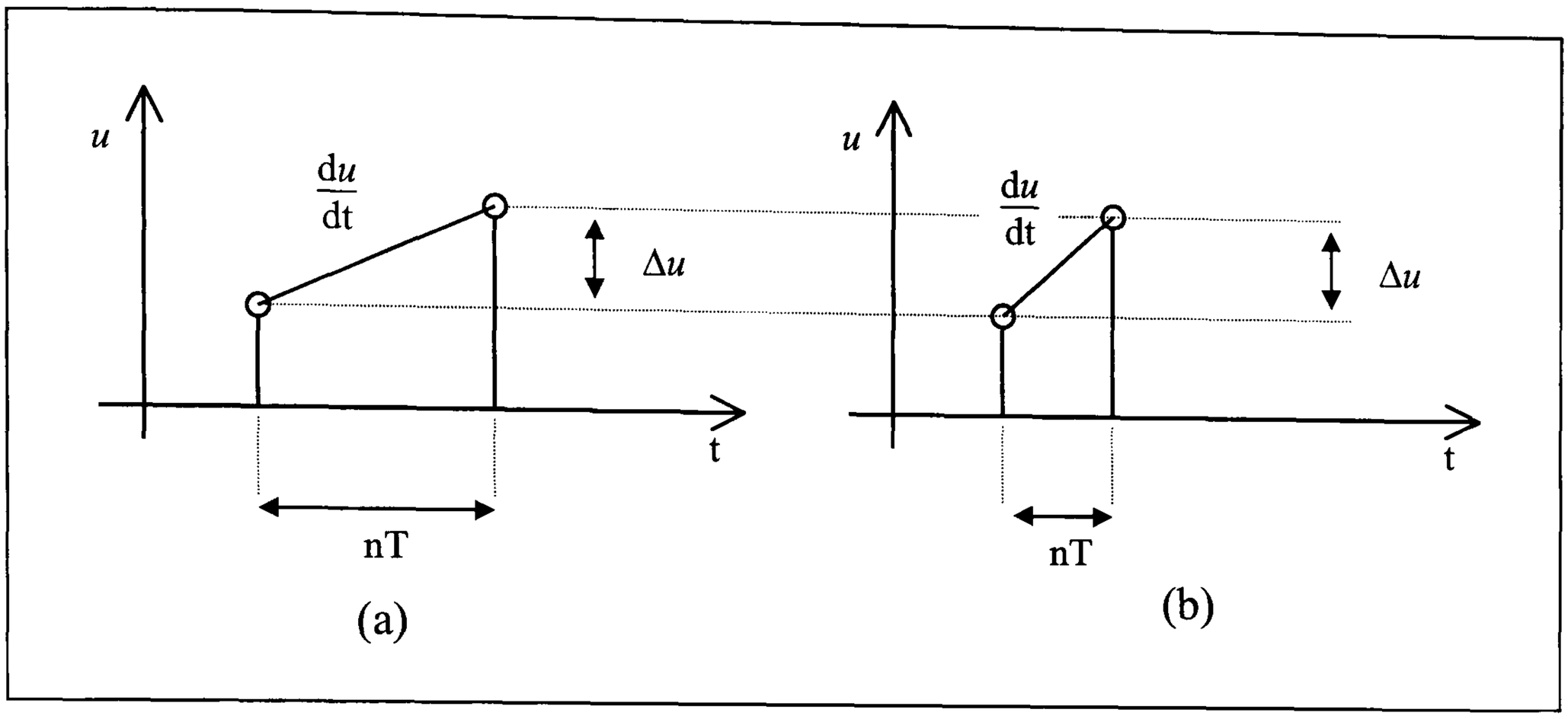


Figure 8-14 *Two output signals with different clock frequency: (a) has a higher frequency than (b).*

A set of tests were carried out to study the response of the controller to different clock frequencies and ultimately select a suitable clock frequency for subsequent experiments. *Figure 8-15* shows a block diagram of the hardware configuration used for this test. The engine-generator set is connected to a bridge rectifier which is loaded with resistive bank sets at 119Ω. Instead of using a crystal oscillator circuit, a function generator is employed to provide a variable frequency clock signal for the electronic control system.

In all the tests, the resistive load is fixed at 119Ω and the desired d.c. voltage is set at 250V. Quantitative discussions are based on numerical data of the test results which are included in *Appendix E* for reference.

Figure 8-16 shows a graphical representation of the results of a test in which the clock frequency is set to 2.0kHz. When the controller is connected at t = 10s, it attempts to bring the d.c. voltage to the desired level but the response is clearly oscillatory with an overshoot of up to 50%. When the clock frequency is set to 0.8kHz, the controller is too slow in its response and results in the engine stalling.

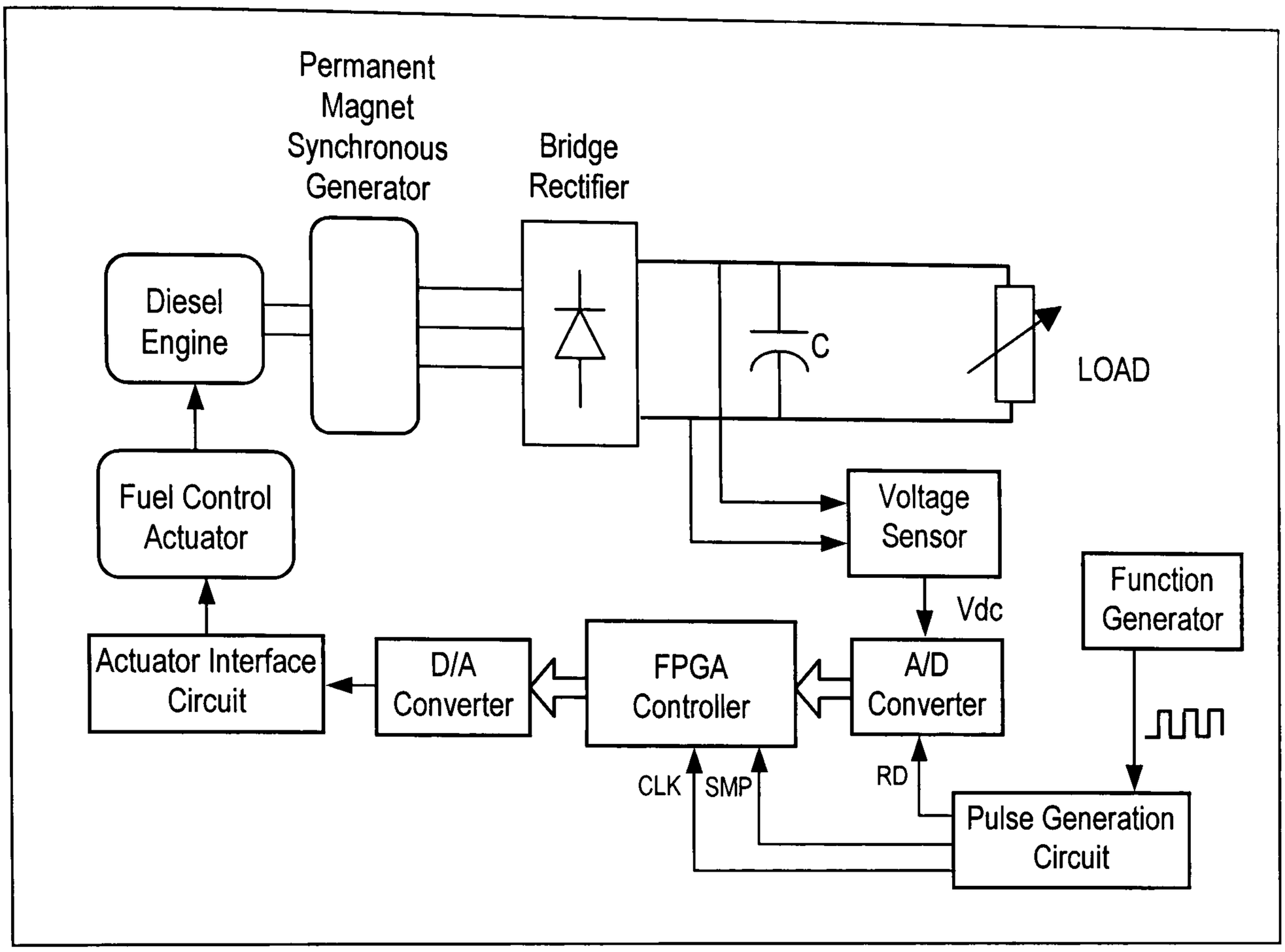


Figure 8-15 *Block diagram of configuration for selecting clock frequency.*

It is found that the best response is obtained with a clock frequency of 1.2kHz. The result is depicted in *Figure 8-17*. Here, the controller is connected at the moment of t = 10s, resulting in an overshoot of not more than 11% but the voltage quickly recovers (in less than 7 seconds compared to 35 seconds in the earlier example) to the desired level demonstrating that the correct actuator position has been set. Although there is a steady state error of about 6.7% to 7.2%, this is allowed for by the design because voltage discrepancies can be compensated for by a feedback inverter control.

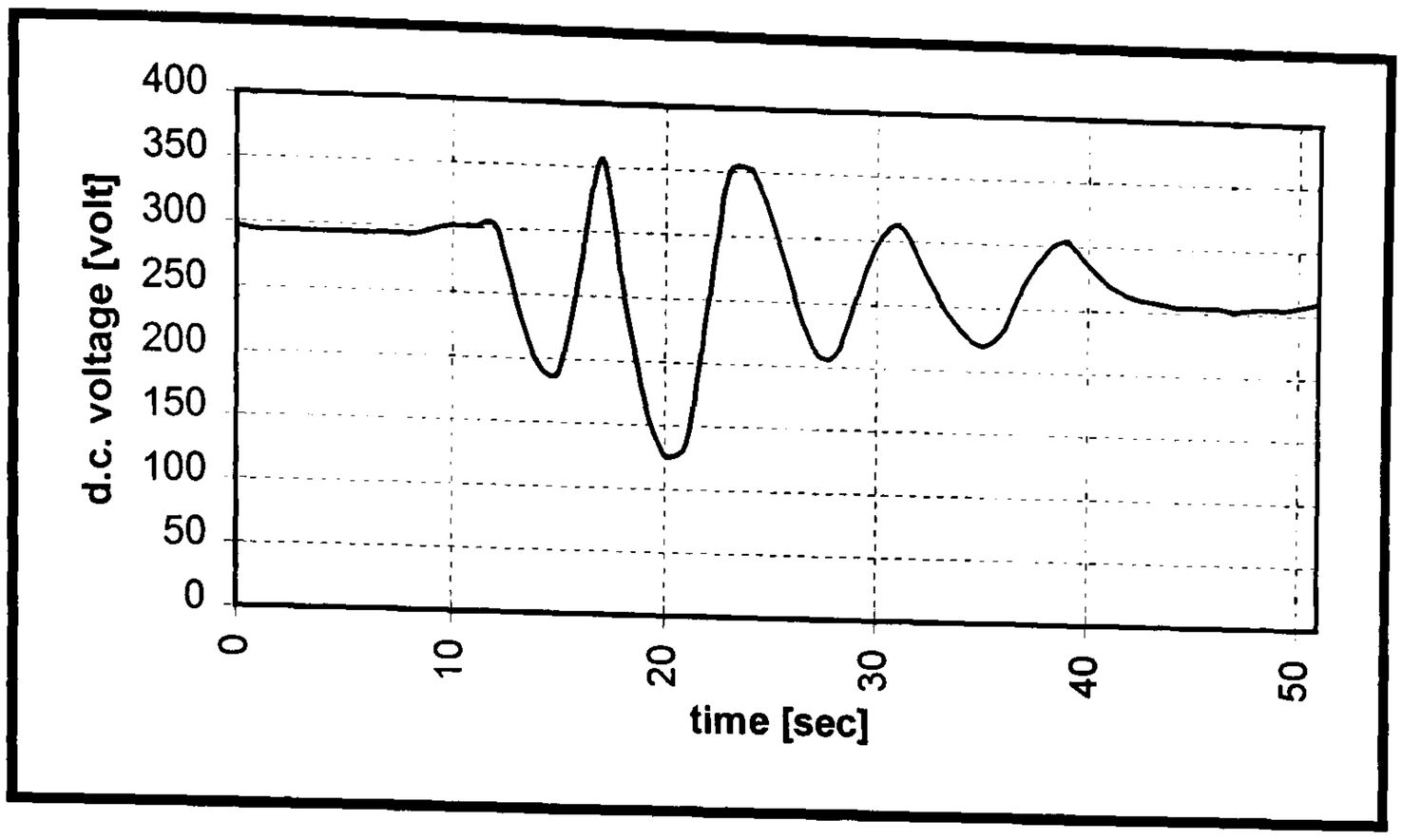


Figure 8-16 *Voltage response with a high clock frequency.*

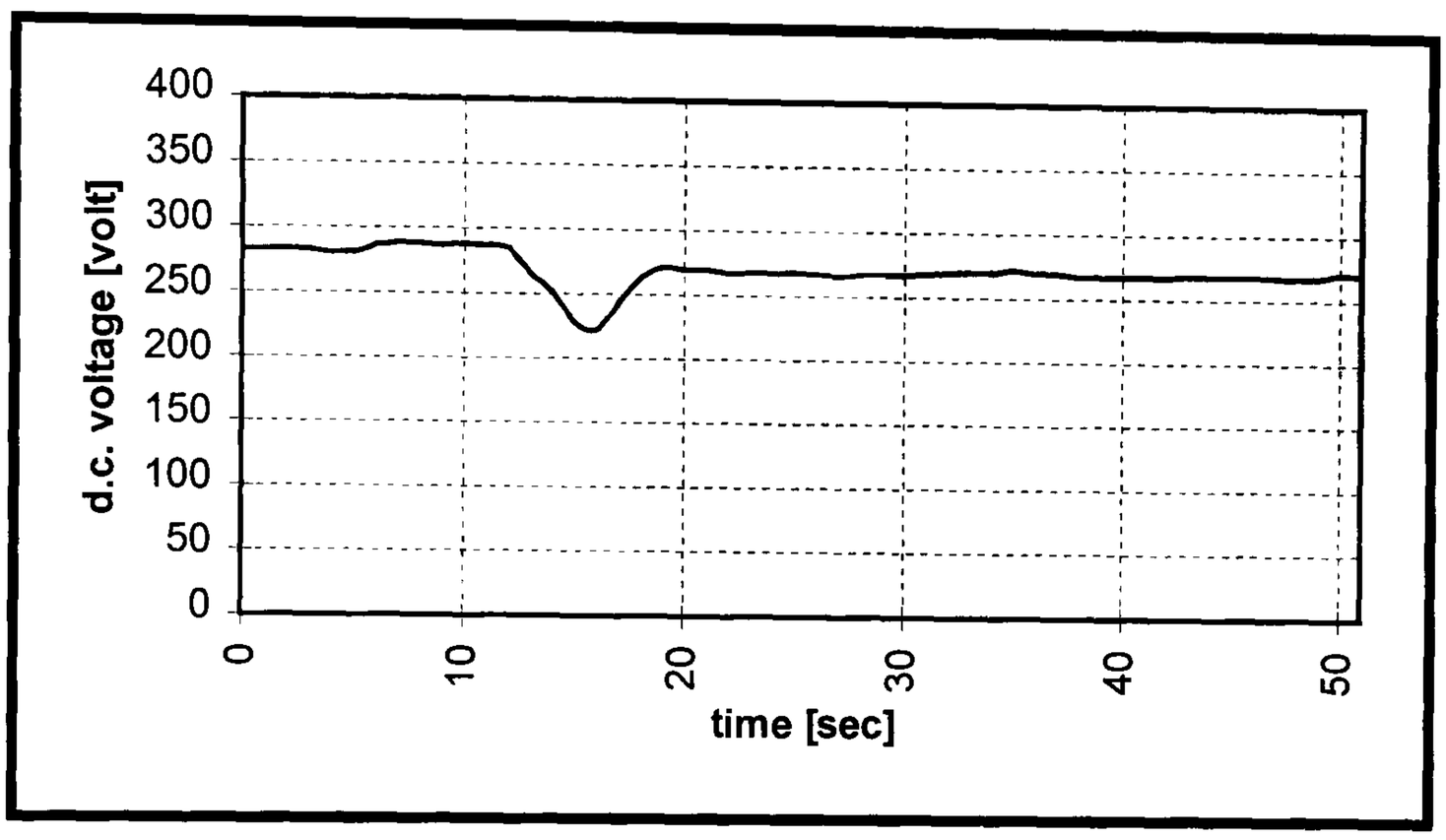


Figure 8-17 *Voltage response with an optimum clock frequency.*

8.4.2 Step change in rectifier load

With a clock frequency of 1.2kHz and the same configuration as before, the rectifier load is now changed from 119.0Ω to 37.5Ω to observe the corresponding change in the system. The load alteration is carried out when t = 20s. For the purpose of comparison, the experiment is first carried out without the control system, whereby the actuator position is fixed. The result, depicted in *Figure 8-18* shows the d.c. voltage gradually dropping to 0V when the load is changed. At the same time, the operational speed is observed to slow down until the engine finally stalls. This experiment confirms that the system is not intrinsically stable under the given change in load.

Figure 8-19 shows the result of the same experiment but with the control system connected. This time the d.c. voltage is maintained close to the desired level although

there is an overshoot of 14% and a small steady state error of about 1.6%. Since it has been demonstrated that the plant cannot cope with the load change on its own, it can be concluded that the control system is successful at maintaining the stability of the plant and the d.c. voltage at the desired level.

It is seen in *Figure 8-18* that the plant cannot cope with the changing load condition. However, when the controller is connected, the output d.c. voltage of the generator is successfully controlled (*Figure 8-19*) and stability is maintained.

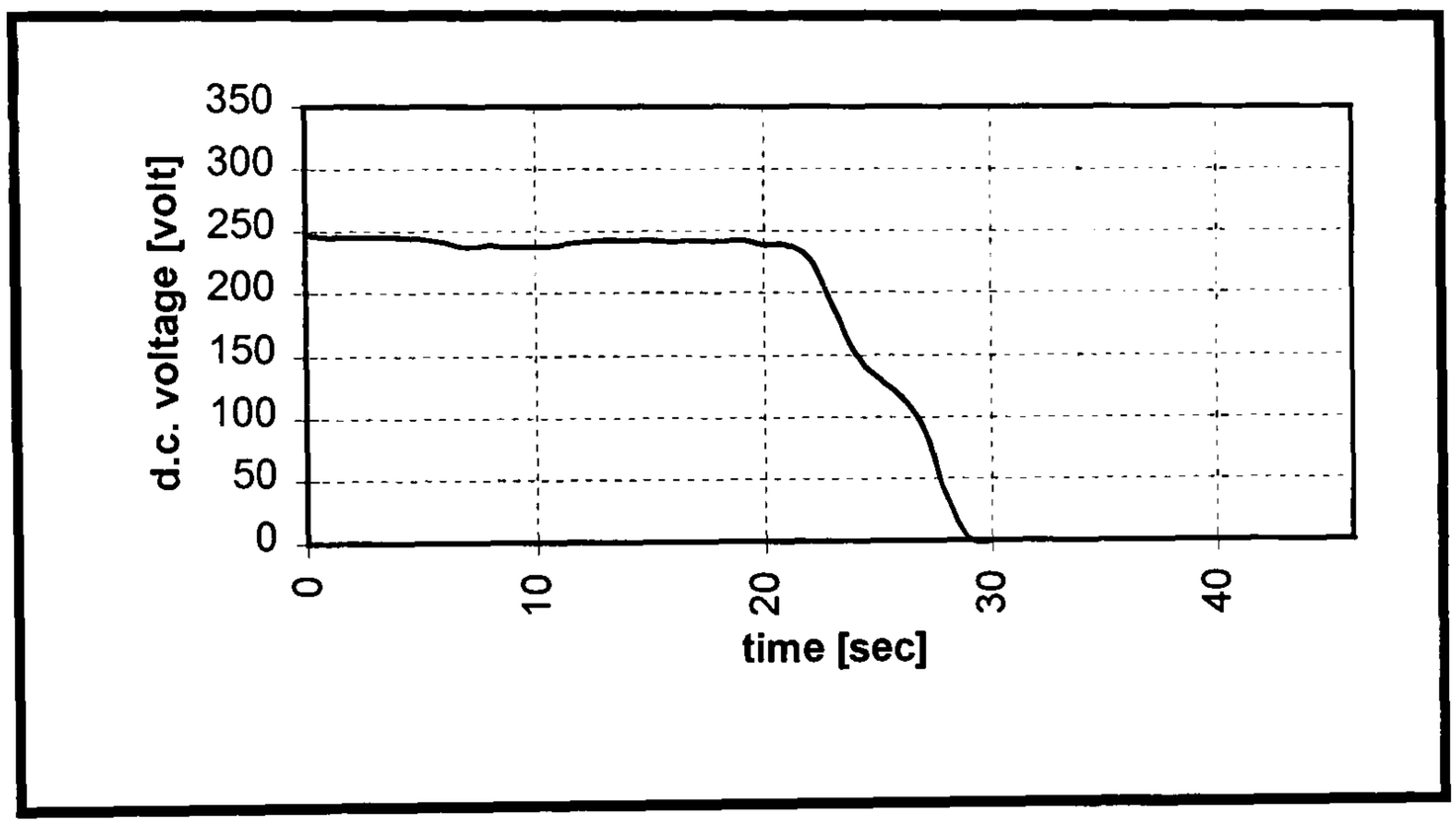


Figure 8-18 *Voltage response without the control system.*

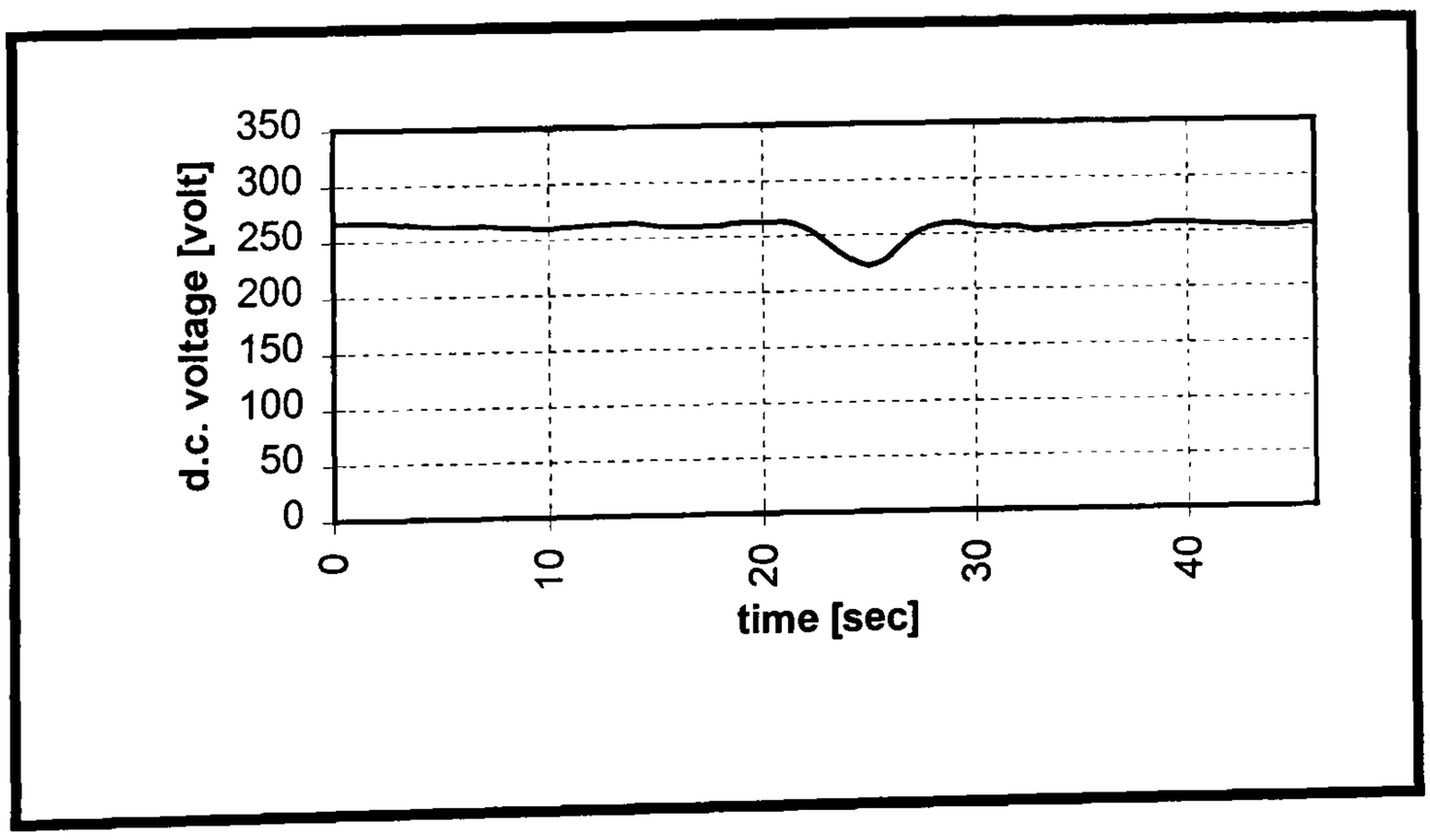


Figure 8-19 *Voltage response with the control system.*

8.4.3 Test configuration with inverter

A block diagram of the complete test configuration including an inverter load is shown at the beginning of the chapter but is included here as *Figure 8-20* for reference. A star-connected variable resistive load is connected to the inverter output. The load resistance is adjusted to obtain a controlled load current of up to 5A (measured at the d.c. link). The measurement of the load current is obtained using a Hall-effect current probe placed on the d.c. link. As before, the desired d.c. voltage is 250V and the clock frequency is set to1.2kHz.

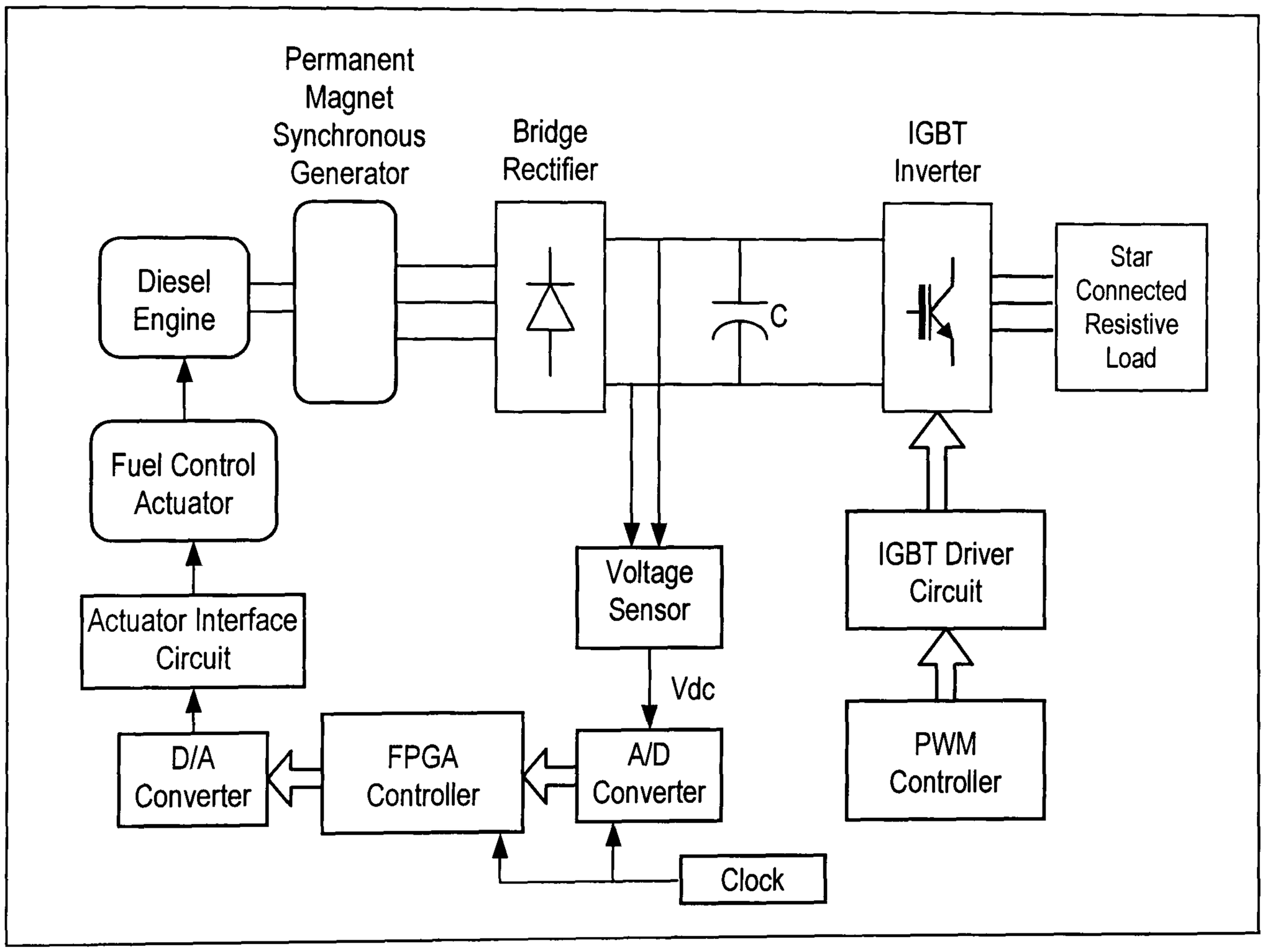


Figure 8-20 *Block diagram of engine generator set and controller.*

Figure 8-21 shows the response when the load current is changed from 0A to 5A at the instant t = 10s. The result is similar to that of *Figure 8-19* in that the d.c. voltage is maintained at the desired level. Following the load change, there is a voltage dip at the d.c. link which settles after 10 seconds. As the controller reacts to compensate for the sudden demand, the d.c. voltage recovers and stabilises at about 250V.

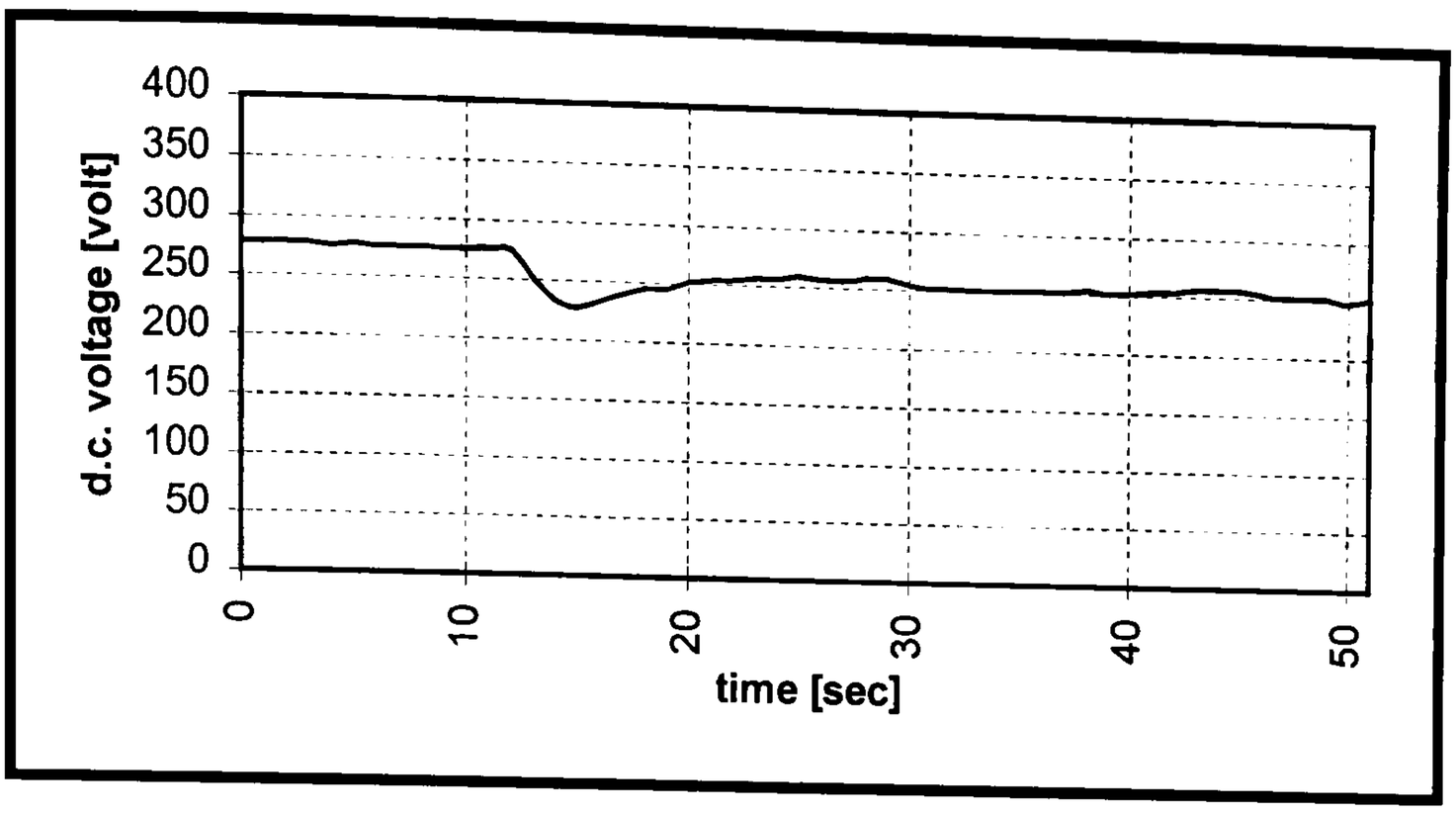


Figure 8-21 *Voltage response with inverter load.*

The materials presented in this chapter show that the design concept is validated and is successfully implemented in hardware. The power electronic system that was developed performed the necessary power conversion while the electronic system achieved the required control effect. The test results shows that the FLC is capable of maintaining the d.c. link voltage within a specified range over a range of changing load conditions. Since it is possible for a PWM inverter to supply a high quality a.c supply (with good frequency control) when the d.c. link voltage is reasonably constant (the exact range of tolerance will depend on the design of the PWM inverter and its control system), the system presented in this thesis is a positive contribution towards the development of a highly efficient and reliable generation of stand alone generator sets that are suitable for sensitive load. The next chapter presents some conclusions on the work as well as further developments to the system which are being investigated.

Chapter
9

Conclusions and Further Work

The conclusions, including some discussion on the present work is presented under two main headings: the control strategy and the design process. The former summarises the discussion relating to variable speed operation and the FLC while the latter section presents some conclusions on the use of VHDL and EDA in designing the hardware.

9.1 The Control Strategy

The control strategy adopted for this research project aimed to enable the operation of the generator at variable speed without disrupting the output frequency. The arguments presented in *Chapter 1* and *Chapter 2* suggested that there are potential rewards if variable speed systems can be controlled to supply constant frequency and stable voltage output. In order to achieve this objective, various power electronic configurations have been considered. A d.c. link converter was considered a suitable solution for two main reasons: the system has been tried and tested in various other applications, particularly in variable speed drives and wind energy conversion, and therefore adopting this approach reduces the risk of unnecessary complications; secondly, the d.c. link converter offers wide flexibility. In addition, various control techniques can be applied to the system, depending on the design criteria.

In the first phase of the project, a d.c. link converter was designed, built and tested. The SA828 IC which is used to control the PWM inverter eliminates the need to design a comprehensive PWM control system from scratch. It also allows the PWM parameters such as the frequency and amplitude modulation ratios to be changed during operation. This is a useful feature for closed-loop control of the output power waveform. The results of the practical tests show that the complete d.c. link converter system is capable of isolating the generator terminal frequency from the final output frequency and that the desired electrical frequency under variable speed operation in a stand alone generator set can be maintained. Consequently, further developments can be approached with confidence.

A Fuzzy Logic Controller (FLC) is designed and built to control the system. Fuzzy logic is found to be suitable for this application because it is able to cope with the non-linearity of the engine-generator system. Furthermore, FLCs are based on linguistic rules and not on mathematical models of the plant, which in this case, can be relatively complex. The FLC is designed in two stages. The first stage is the design of a generic double-input, single-output FLC. In the second stage, the FLC is incorporated with the appropriate interfacing blocks, knowledge base and rule base to give it a very specific functionality. This approach allows the functionality of the FLC to be changed simply by redefining the parameters in the second stage without the need to redesign its basic structure. In the experiments presented in the thesis, the FLC is designed to regulate the d.c. link voltage within specified values using a set of 25 fuzzy rules. The rules are derived from the characteristics of a PI controller and executed using Mamdani's inference technique. The technique relies on straight forward *min* and *max* operations which can easily be implemented. This thesis also proposed an original method of modelling the inference engine, whereby the relation between the rule-consequence and output values is represented by a set of switches. By using this model, the designer is supplied with a clear and well-defined structure of a variable rule base. The collective effect of the rules are determined by the ON/OFF status of the switches. Currently, the output voltage of synchronous generators is usually controlled using Automatic Voltage Regulators (AVRs). AVRs operate by using the generator's field current to regulate the output voltage. This is widely recognised as an effective technique for the control of

electromagnet synchronous generators. However, conventional AVRs cannot be used to regulate the output voltage due to the absence of field windings. In this thesis a controller capable of controlling both electromagnetic and permanent magnet generators has been presented. The practical tests of the system were conducted on a 7.5kW permanent magnet synchronous generator. The control strategy is effective in controlling the permanent magnet generator reliably. This is a progressive contribution towards the development of high performance permanent magnet generator sets.

From the discussion, the following conclusions about the control strategy adopted can be drawn:

- Variable speed operation of a stand alone synchronous generator is feasible control strategy from the point of view of design and implementation. This is possible due to the advance in power electronic technology which offers a relatively cheap and effective solution to maintaining a constant output frequency under variable speed operation.

- The use of fuzzy logic in the present control application improves the performance while offering greater flexibility. The performance of the controller is good because it is not limited by the accuracy of the plant model and can incorporate human experience of the plant in the form of a linguistic rule base. Flexibility is increased by the generic nature of the basic FLC structure. The way in which the FLC is designed allows for easy implementation into hardware as well as easy manipulation of its functionality.

- The method of generator control presented in this thesis is not only suitable for electromagnet generators but can also be used to control permanent magnet generators. This is because the system is not dependent on the field current for voltage regulation.

- The complete system shows encouraging results in controlling a diesel engine driven synchronous generator operating under variable load and speed. It is particularly suitable for studying the specific benefits of variable speed generators such as reduced noise, vibration and fuel consumption.

9.2 The Design Process

Although VHDL is essentially a hardware design language, the work in this thesis has explored the use of VHDL in mathematical modelling and simulation of control strategies. From this experiment, it can be concluded that the approach offers substantial advantages in the design and development process of a control system, specifically if the control system is to be implemented into FPGAs or ASICs.

The fact that VHDL is a component-oriented language makes it easy to create reusable models of the control plant. In the project, the generator and rectifier are modelled in VHDL and configured as a single component *Genrect*. A model of the diesel engine is also described in VHDL and configured as the component *Engine*. The two components are then combined together to create a complete model of a stand-alone generator set which is connected to a rectifier and because these components can be used individually or connected together to form larger systems, they are very effective for computer verification and simulation of control strategies. It is possible to build libraries with models of generators, engines and motors for such purposes.

The entire design process of the FLC is achieved using VHDL. It is found that by writing design, simulation and implementation codes in the same language, there is a significant reduction in the concept-to-implementation time frame. In a conventional design process, separate programming codes have to be written for simulation and implementation purposes. For example, if a design is to be verified using MATLAB and implemented using a microprocessor, the MATLAB codes used for the simulations have to be translated into assembly language for implementation. The translation process can give rise to errors which may not be easily detected. This complication has been successfully avoided in the work presented here. Another advantage of using VHDL is that the design can be kept relatively independent of the implementation technology for most parts of the design process. It is only in the final stages that the FLC design in this thesis is targeted for the Xilinx XC4010 FPGA. Unlike microprocessor and DSP codes, the VHDL design can be recycled and used for implementation into different types of FPGAs and ASICs.

In summarising the conclusions on the design process:

- VHDL can be effectively used as a modelling and simulation tool in the design control systems.

- Using the same language for simulation and implementation eliminates the need for translation and therefore reduces the possible errors in the design. This approach also reduces the concept-to-implementation time frame which is a very important advantage under the present market demands.

- VHDL designs have the advantage of being relatively technology-independent. Therefore, the designs can be updated and recycled in line with the advances in hardware technology.

- Optimisation is an essential part of FPGA design. Various methods can be used to reduce significantly the area space of a design but they usually come at the expense of performance.

This thesis has presented the design from concept through to the implementation of a fully functional prototype intelligent controller for a stand-alone synchronous generator set. The control system has been successfully tested on a permanent magnet generator coupled to a diesel engine. In the course of the work, several original ideas were developed. The use of VHDL to model and simulate the control strategy made it considerably easier to verify the design before implementation. By introducing a method of modelling and designing the fuzzy inference engine using *Mini*-FAM tables, it was possible to reduce the area size of the FLC's design. Numerous analogue, digital as well as power electronic circuits have been designed and constructed to implement the proposed control strategy.

The VSCF stand alone generator system is an approach that can improve the efficiency and reliability of the system. Using fuzzy logic in the implementation of the control strategy ensures that the system is sufficiently flexible and effective while the use of VHDL and a common platform for simulation and implementation means that the design can be upgraded easily and rapidly. From the outcome of the research, it can be concluded that all the original objectives have been met.

9.3 Further Work

This section discusses some possible improvements to the system and areas which warrant further investigation. Two developments to the system have been considered.

The first area involves the D.C.-A.C. conversion, i.e. the PWM inverter. A closed loop control system as well as a suitable filtering system would ensure a high quality electricity supply to the load. Although the function of the FLC is to maintain the d.c. link voltage at a specified value, the large delays in the engine and generator means that the response is relatively slow. To compensate for this, a closed loop control system for the PWM inverter which takes into account the variation in the d.c. link voltage would be able to provide a faster and more accurate regulation of the output voltage. The basic aspects of this type of control system were analysed and discussed in *Chapter 5*. However, a more comprehensive study has already been initiated to investigate the PWM control strategies [94] and to ultimately design a high performance control system based on the present work.

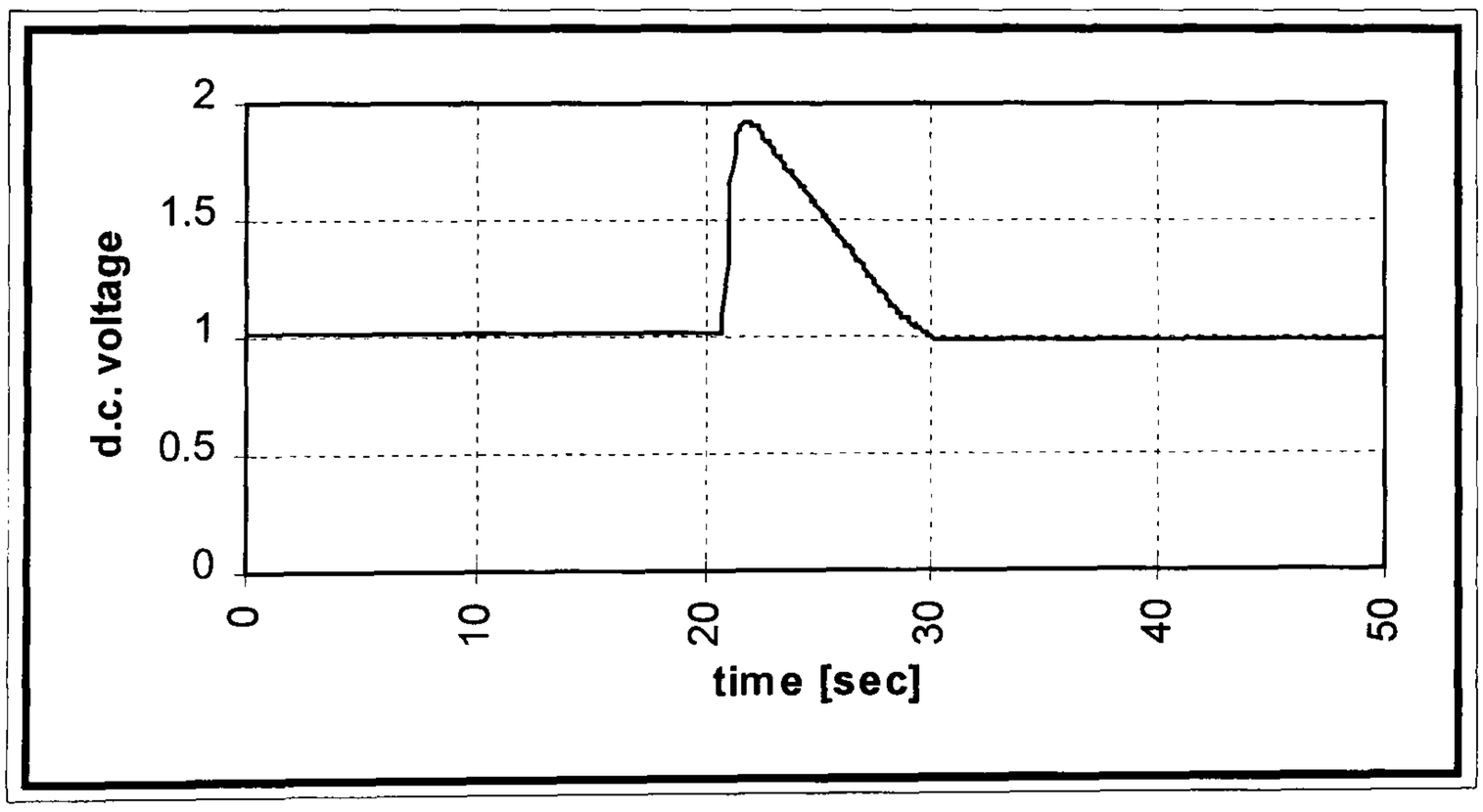


Figure 9-1 *Voltage response when there is a large and sudden load variation.*

Another important issue which needs to be addressed is the effect of large and sudden changes in the load. This may result in voltage transients of large magnitudes, creating serious problems in the system such as stress on the power electronic devices. *Figure 9-1* shows the result of a simulation whereby the load current is reduced from 25A to 1A. It can be observed that the magnitude of the voltage surge caused by the shedding of load is almost twice its reference value. The reason for this is that the overall response

of the present system is too slow to cope with the transient conditions. A possible solution to this problem is to introduce a D.C.-D.C. conversion stage to the system like the one shown in *Figure 9-2*.

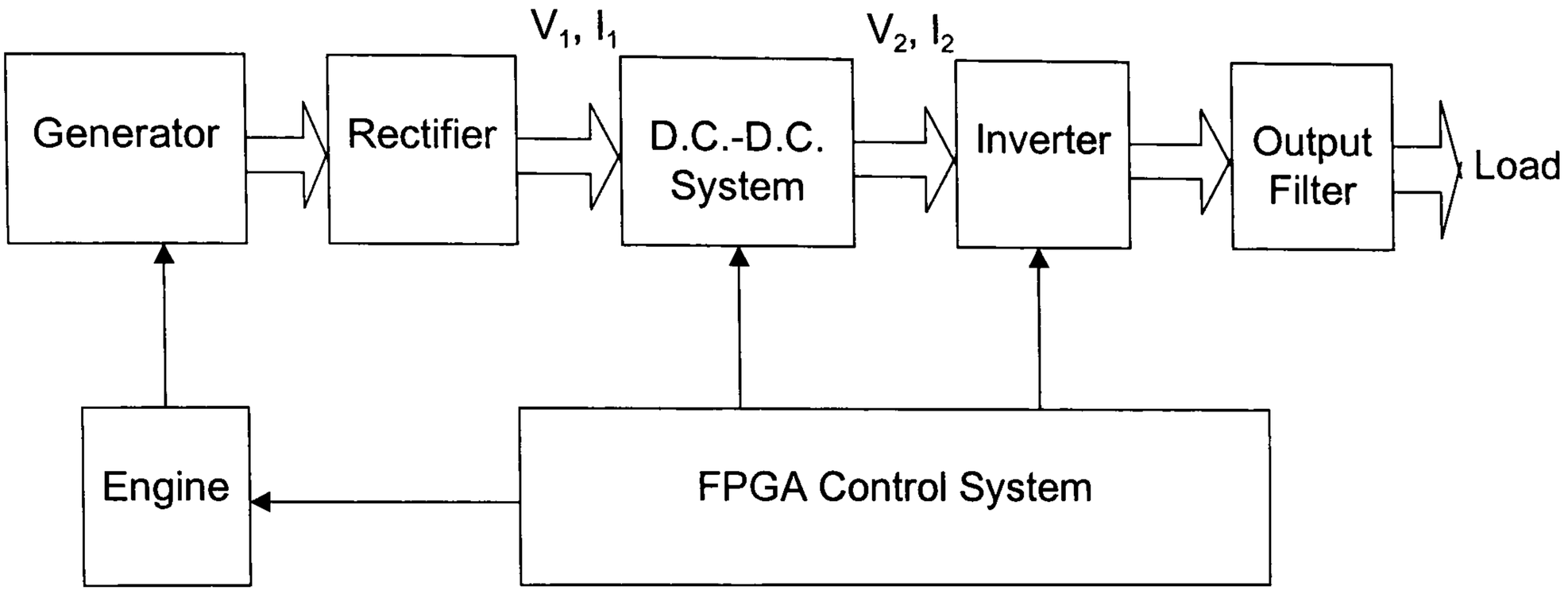


Figure 9-2 *Possible improvements to the present system.*

The requirements of the D.C.-D.C. system are:

- able to absorb transient voltage surge
- able to compensate for transient voltage dips
- cost effective.

This can be achieved using a battery coupled to an appropriate power electronic and control system. The aim is to make up for the difference in the energy supply and demand. Once all the practical issues have been properly addressed, the system is anticipated to be a robust and highly efficient power generator that is suitable for sensitive loads.

There are also substantial advantages to be gained from the use of VHDL as a modelling and simulation tool in the design of power electronic control systems. The full potential of this approach can only be realised with the development of VHDL libraries that contains comprehensive models of electrical machines, power converters, engines and other common equipment.

PAGE NUMBERING AS ORIGINAL

REFERENCES

[1] Binns, K.J., Kurdali, A. *'Permanent Magnet A.C. Generator.'* Proc. IEE vol. 126, No. 7, Jul 1979. pp 690-696.

[2] Rahman, M.A., Osheiba, A.M., Radwan, T.S. *'Modelling and Controller Design of An Isolated Diesel Engine Permanent Magnet Synchronous Generator.'* IEEE Trans. on Energy Conversion, Vol.11, No. 2, June 1996. pp 324-329.

[3] Akmese, R., Chalmers, B.J. *'Permanent Magnet Alternators with Constant Output Voltage.'* Proc. 37th Universities Power Eng. Conf. UPEC'97, vol. 1, 10-12th Sept. 1997. pp34-36.

[4] De Mello, F.P., Hannet, L.N. *'Large Scale Induction Generators For Power Systems.'* IEEE Trans. on Powser Apparatus & Systems, PAS 100, 1981. pp 2610-2618.

[5] Chan, T.F. *'Self-Excited Induction Generators Driven By Regulated And Unregulated Turbines.'* IEEE Trans. on Energy Conversion, Vol. 11, No. 3, Sept 1996. pp 338-343.

[6] Alolah, A.I. *'Steady State Operating Limits Of Three Phase Self-Excited Reluctance Generator.'* IEE Proc. C, Vol. 139, No. 3, May 1992. pp 261-268.

[7] Adler, M.S., et. al. *'The Evolution of Power Device Technology.'* IEEE Trans. Electron Devices, vol. 31, no. 11, Nov. 1984. pp 1570-1591.

[8] Baliga, B.J., et. al. *'The Insulated Gate Transistor (IGT) - A New Power Switching Device'* IEEE/IAS Ann. Meet. Conf. Rec., 1983. pp 794-803.

[9] Scharf, A. *'Twenty Years of Innovation.'* PCIM Magazine, issue 6, 1999. pp10-12.

[10] Sankara Narayanan, E.M., De Souza, M.M., Qin, Z. *'Devices and technologies for High Voltage Integrated Circuits'*, Proc. Int. Conf. on Semiconductor Materials and Technologies, New Delhi, India,16-21 Dec.1996.

[11] Thomas, M.J. *'Designing Intelligent Muscle into Industrial Motion Control.'* IEEE Trans. Ind. Electron., vol. 37, no. 5, 1990. pp 329-341.

[12] Maxwell, J.C. *'On Governors.'* Proc. Roy. Soc. (London) , 1868.

[13] Hazen, H.L. *'Theory of Servomechanism.'* J. Franklin Inst., 1934.

[14] Evans, W.R. *'Graphical Analysis of Control Systems.'* Trans. AIEE, 1948.

[15] Misawa, E.A., Hedrick, J.K. '*Non-linear Observers - A State of the Art Survey.*' ASME J. Dyn. Syst. Meas. Contr., vol. 111, 1989. pp344-352.

[16] Cilia, J., Asher, G.M., Bradley, K.J., '*Sensorless Position Detection for Vector Controlled Induction Motor Drives Using and Asymmetric Outer-Section Cage.*' IEEE Trans. on IAS, vol. 33, no. 5, Sept/Oct 1997. pp 1162-1169.

[17] Ames, Robert L. 'A.C Generators : Design & Application.' Research Studies Press, UK, 1990. Chapter 9.

[18] Weedy, B.M. 'Electric Power Systems.' John Wiley & Sons, Third Edition, 1995. Chapter 3.

[19] Newage International Ltd. Technical Reference Manual, Section 4131, Jan 1990. p1.

[20] Malik, O.P., Hope, G.S., Huber, D.W. '*Design And Test Results Of A Software Based Digital A.V.R .*' IEEE Trans. on Power Appliances and Systems, Vol. PAS-92, no. 2, Mar/Apr 1976. pp 634-642.

[21] Huber, D.W., Runtz, K.J., Malik, O.P., Hope, G.S. '*Analytical And Experimental Studies Of A Digital A.V.R .*' IEEE 8th PICA Conf. Proc. 1973. pp 195-203.

[22] Hirayama, K., et. al. '*Digital A.V.R. Application to Power Plants.*' IEEE Trans. on Energy Conversion, Vol. 8, No. 4, Dec. 1993.

[23] Godhwani, A., Basler, M.J. '*A Digital Excitation Control System For Use On Brushless Excited Synchronous Generators.*' IEEE Trans. on Energy Conversion, Vol. 11, No. 3, Sept 1996. pp 616-619.

[24] Marino, R. '*An example of Non-linear Regulator.*' IEEE Trans. Automatic Contr., vol. 29, no. 3, 1984. pp276-279.

[25] Savaresi, S.M. '*Exact Feedback Linearisation of a Fifth Order Model of Synchronous Generators.*' IEE Proc. Contr. Theory Appl., vol. 146, no. 1, Jan. 1999. pp53-57.

[26] Ibrahim, A.S., Hogg, B.W., Sharaf, M.M. '*Self-tuning Automatic Voltage Regulators for a Synchronous Generator.*' IEE Proc. Pt. D, vol. 136, no. 5, Sept. 1989. pp252-260.

[27] Farsi, M., Zachariah, K.J., Finch, J.W. '*Implementation of a Self-tuning AVR.*' IEE Proc. - Control Theory Appl., vol. 144, no. 1, Jan. 1997. pp32-39.

[28] Wu, Q.H., Hogg, B.H. '*Robust Self-tuning Regulator for a Synchronous Generator.*' IEE Proc. Pt. D, vol. 135, no. 6, Nov. 1988. pp463-473.

[29] Morioka, Y., et. al. '*Application Of Multivariable Optimal Controller to Real Power Systems.*' IEEE Trans. on Power System, vol. 9, no. 4., 1994. pp 1949-1955.

[30] Ghosh, A., Ledwich, G., Malik, O.P., Hope, G.S. '*Power Systems Stabilisers Based On Adaptive Control Techniques.*' IEE Trans. on Power Apparatus and Systems, vol. PAS-103, no.8, 1984. pp 1983-1989.

[31] Pahalawaththa, N.C., Hope, G.S., Malik, O.P. '*Multivariable Self Tuning Power System Stabiliser Simulation and Implementation Studies.*' IEEE Trans. on Energy Conversion, vol. 6, no. 2, 1991. pp 310-319.

[32] Hirayama, T. '*Robustness of Fuzzy Logic Power System Stabilisers Applied to Multimachine Power System.*' IEEE Trans. on Energy Conversion, vol. 9, no. 3, 1994. pp 451-459.

[33] Kennedy, D.C., Quintana, V.H. '*Neural Network Regulators For Synchronous Machines.*' Proc. of ESAP 1993. pp 531-535.

[34] Zhang, Y., Malik, O.P., Chen G.P. '*Artificial Neural Network Power System Stabilisers in Multimachine Power System Environment.*' IEEE Tran. on Energy Conversion, vol. 10, no. 1, 1995. pp 147-153.

[35] Kobayashi, T., Yokoyama, A. '*An Adaptive Neuro-Control System Of Synchronous Generator for Power System Stabilisation.*' IEEE Trans. on Energy Conversion, vol. 11, no. 3, Sept 1996. pp 621-630.

[36] Venturini, M.G.B., Alesina, A. '*A New Sine Wave In, Sine Wave Out, Conversion Technique Eliminates Reactive Elements.*' Proc. Powercon 7, 1980. ppE3.1-E3.15.

[37] Zhang, L., Watthanasarn, C., Shepherd, W. '*Analysis and Comparison of Control Techniques for A.C.-A.C. Matrix Converters.*' IEE Proc. Electr. Power Appl., vol. 145, no. 4, July 1998. pp284-294.

[38] Wheeler, P.W., Grant, D.A. '*A Low Loss Matrix Converter for A.C. Variable Speed Drives.*' Proc. EPE'93, U.K., vol. 5, 1993. pp27-32.

[39] Tenti, P., Malesani, L., Rossetto, L. '*Optimum Control of N-input K-output Matrix Converters.*' IEEE Trans. Power Electron., vol. 7, no. 4, 1992. pp707-713.

[40] Bleijs, J.A.M., Wong, K.J. '*Control of a Variable Speed Wind Turbine with a Synchronous Generator and Boost Rectifier.*' Proc. 37th Universities Power Eng. Conf. UPEC'97, 10-12th Sept. 1997. pp722-725.

[41] Pena, R.S., Asher, G.M., Clare, J.C. '*A Doubly-fed Induction Generator Using Back-to-back PWM Converters Supplying an Isolated Load from a Varibale Speed Wind Turbine.*' IEE Proc. Pt. B, vol. 143, no. 5, Sept. 1996. pp380-387.

[42] Hilloowala, R.M., Sharaf, A.M. '*A Rule-Based Fuzzy Logic Controller for a PWM Inverter in a Stand Alone Wind Energy Conversion Scheme*' IEEE Trans. on Industry Appl., vol. 32, no. 1, 1996. pp57-65.

[43] Simoes, M.G., Bose, B.K., Spiegel R.J. '*Design and Performance Evaluation of a Fuzzy Logic Based Variable Speed Wind Generation System.*' IEEE Trans. on Industry Appl., vol. 33, no. 4, 1997. pp956-965.

[44] Miller, A., Muljadi, E., Zinger, D.S. '*A Variable Speed Wind Turbine Power Control.*' IEEE Trans. on Energy Conversion, vol. 12, no. 2, 1997. pp181-186.

[45] Bleijs, J.A.M. '*Improvement in Performance of a Passive Pitch Wind Turbine with Variable Speed Operation.*' Proc. of 5th Eur. Wind Energy Assoc. Conf. EWEC, Thessaloniki, Greece, vol. 1, Oct. 1994. pp588-592.

[46] Jones, R., Smith, G.A. '*High Quality Mains Power Form Variable Speed Wind Turbines.*' Int. Conf. on Renewable Energy - Clean Power 2001, IEE Conf. Publ. No. 385, 17-19 Nov. 1993. pp 202-206.

[47] Smith, G.A., Stephens, R.G., Marshall, P., Kansara, M. '*A Sinewave Interface For Variable Speed Wind Turbines.*' 5th European Conf. on Power Electronics & Applications, IEE Conf. Publ. No. 377, 13-16 Sept. 1993. pp 97-102.

[48] Chen, Z., Spooner, E., '*Grid Interface Options for Variable Speed Permanent Magnet Generators.*' IEE Proc. Electr. Power Appl., vol. 145, no. 4, July 1998. pp273-282.

[49] Blayliss, John. '*Electronic Design Automation Report.*' Cambridge Market Intelligence, 1994.

[50] Patterson, E.B. '*Electronic Design Automation For Power Electronic Drives.*' PhD Thesis, Nottingham Trent University, 1993.

[51] Patterson, E.B. ; Holmes, P.G. ; Morley, D. '*Electronic Design Automation (EDA) Techniques For The Design Of Power Electronic Control Systems.*' IEE Proc. - G, Vol. 139, No.2, April 1992. pp 191-198.

[52] Cirstea, M.N., Patterson, E.B., Morley, D. '*The Development And Design Of An Universal Digital Control System For Cycloconverter Drives Using Electronic Design Automation (EDA) Techniques.*' Proc. UPEC'95, Greenwich, UK, 5-7 Sept. 1995. pp725-728.

[53] IEEE VHDL 1076-1987 *Language Reference Manual*, Dec. 1987.

[54] Perry, D.L. 'VHDL' McGraw Hill, Second Edition, 1993.

[55] Sjoholm, S., Lindh, L. 'VHDL for Designers' Prentice Hall, 1997.

[56] 'Foundation Series Quick Start Guide 1.4' Xilinx Inc. 1991-1997.

[57] den Bout, D.V. 'The Practical Xilinx Designer Lab Book' Prentice Hall, 1998.

[58] Sen, P.C. 'Principles of Electric Machines and Power Electronics.' John Wiley & Sons, Second Edition, 1997.

[59] Boost, M.A. ; Ziogas, P.D. *'State-of-the-Art Carrier PWM Techniques : A Critical Evaluation.'* IEEE Trans. Ind. Applicat., Vol. 24, No. 24, Mar/Apr 1988. pp 271-280.

[60] Hui, S.Y.R. ; Oppermann, I. ; Sathiakumar, S. *'Microprocessor-Based Random PWM Schemes For D.C.-A.C. Power Conversion.'* IEEE Trans. on Power Electronics, Vol. 12, No. 2, Mar 1997. pp 253-260.

[61] Habetler, T.G. ; Divan, D.M. *'Acoustic Noise Reduction In Sinusoidal PWM Drives Using A Randomly Modulated Carrier.'* IEEE Trans. on Power Electronics, Vol. 6, July 1991. pp 356-363.

[62] Jung, S.L. ; Tzou. Y.Y. *'Discrete Sliding Mode Control Of A PWM Inverter For Sinusoidal Output Waveform Synthesis With Optimal Sliding Curve.'* IEEE Trans. on Power Electronics, Vol. 11, No. 4, July 1996. pp 567-577.

[63] Mohan, N., Underland, T.M., Robbins, W.P., 'Power Electronics: Converters Applications and Design' John Wiley & Sons, 1989.

[64] GEC Plessey SA828 Data Sheet.

[65] Zadeh, L.A. '*Outline of a new approach to the analysis of complex systems and decision processes.*' IEEE Trans. Syst., Man & Cybern., vol.3, 1973. pp28-44.

[66] Zadeh, L.A. '*Fuzzy Sets*' Information and Control, vol. 8, 1965. pp338-353.

[67] Smith, F.S., Shen, Q. '*Selecting Inference and Defuzzification Techniques for Fuzzy Logic Control.*' Proc. of UKACC International Conference on Control, 1-4 Sept. 1998. pp54-59.

[68] Ross, R.J. 'Fuzzy Logic with Engineering Applications.' McGraw-Hill, 1995.

[69] Driankov,D, et. al. 'An Introduction to Fuzzy Control.' Springer-Verlag 1993.

[70] Song, Y, Johns, A.T. '*Applications of fuzzy logic in power systems, Part 1.*' IEE Power Engineering Journal, Oct. 1997.

[71] Song, Y, Johns, A.T. '*Applications of fuzzy logic in power systems, Part 2.*' IEE Power Engineering Journal, Aug. 1998. pp185-190.

[72] Song, Y, Johns, A.T. '*Applications of fuzzy logic in power systems, Part 3.*' IEE Power Engineering Journal, Apr. 1999. pp97-103.

[73] Bose, B.K. '*Expert System, Fuzzy Logic and Neural Network Applications in Power Electronics and Motion Control.*' Proc. IEEE, vol. 82, Aug. 1994. pp1303-1323.

[74] Sousa, G.C.D., Bose, B.K. '*A fuzzy set theory based control of a phase-controlled converter dc machine drive.*' IEEE Trans. Ind. Appl., vol. 30, 1994. pp34-44.

[75] Simoes, M.G., et. al. '*Fuzzy logic based intelligent control of a variable speed cage machine wind generation system.*' IEEE Trans. Power Electronics, vol.12, no.1, Jan 1997. pp. 87-94.

[76] Sousa, G.C.D., et. al. '*Fuzzy logic based on-line efficiency optimisation control of an indirect vector controlled induction motor drive.*' Proc. IEEE/IECON Conf., 1993. pp1168-1174.

[77] Simoes, M.G., Bose, B.K. '*Application of fuzzy logic in the estimation of power electronic waveforms.*' IEEE/IAS Annual Meet. Conf. Rec., 1993. pp 853-861.

[78] Smith F.S., Shen Q. *'Selecting Inference and Defuzzification Techniques for Fuzzy Logic Control.'* Proceedings of UKACC Int. Conf. on CONTROL'98, Sept. 1998. pp54-59.

[79] Smith F.S., Shen Q. *'Choosing the Right Fuzzy Logic Controller.'* Proceedings of 7th Int. Fuzzy Systems Association World Congress, 1997. pp342-347.

[80] Runkler T.A. *'Selection of Appropriate Defuzzification Methods Using Application Specific Properties.'* IEEE Trans. Fuzzy Systems, Feb. 1997. pp72-79.

[81] Karr, C.L., Gentry, E.J. '*Fuzzy Control of pH Using Genetic Algorithms.*' IEEE Trans. Fuzzy Syst., vol. 1, no. 1, Jan. 1993. pp46-53.

[82] Hmaifar, A., McCormick, E. *'Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms.'* IEEE Trans. Fuzzy Syst., vol. 3, no. 2, May 1995. pp129-139.

[83] Hwang, H.S. *'Automatic Design of Fuzzy Rule Base for Modelling and Control Using Evolutionary Programming.'* IEE Proc. Contr. Theory Appl., vol. 146, no. 1, Jan 1999. pp9-16.

[84] Mudi, R.K., Pal, N.R. *'A Robust Self-Tuning Scheme for PI- and PD-Type Fuzzy Controllers.'* IEEE Trans. Fuzzy Systems, vol. 7, no. 1, Feb 1999. pp2-16.

[85] Lee, J. *'On Methods for Improving Performance of PI-Type Fuzzy Logic Controllers.'* IEEE Trans. Fuzzy Systems, vol. 1, no. 4, Nov 1993. pp298-302.

[86] Ogata K. 'Discrete-time Control System.' Prentice-Hall 1995.

[87] Rose, C. *'Masters of their domain.'* New Electronics On Campus, Autumn 1998. p43.

[88] VHDL Standard Library, *ieee.std_logic1164*.

[89] 'The Programmable Logic Data Book.' Xilinx 1998.

[90] Kropf, T. *'Benchmark Circuits for Hardware Verification.'* Universität Karlsruhe, Kaiserstr, Germany. Website http://goethe.ira.uka.de/hvg, accessed Feb 1999.

[91] McCalla, T.R. 'Digital Logic and Computer Design.' Macmillan, 1992.

[92] De Micheli, G. 'Synthesis and Optimisation of Digital Circuits.' McGraw Hill 1994.

[93] *'XS40 and XS95 Board Manual'*, XESS Corp. Website http://www.xess.com/FPGA/ho02001.htm, accessed Oct 1998.

[94] Cirstea, M.N., Khor, J.G., Hu, Y., McCormick, M., Haydock, L. *'Intelligent Fuzzy Logic Controller for Power Generation Systems.'* Proc. 9th Int. Conf. on Electrical Machines and Drives, UK, Sep. 1999. pp321-324.

BIBLIOGRAPHY

[1] Bose, K.B. '*Recent Advances in Power Electronics.*' IEEE Trans. Power Electronics, Vol. 7, No. 1, Jan. 1992. pp2-15.

[2] Cirstea, M.N. '*An Investigation into ASIC Control of 6-Pulse Cycloconverter for A quad Winding Induction Motor.*' PhD Thesis, The Nottingham Trent University, 1996.

[3] Cirstea, M.N., Patterson, E.B., Morley, D. '*A Complete ASIC Controlled Electric Drive System.*' Proc. Int. Symposium for Circuits and Systems ISCAS'96, Atlanta, USA, 12-15 May 1996. pp561-564.

[4] Cirstea, M.N., Patterson, E.B., Morley, D., Holmes, P.G., '*A Universal Digital Control System for Cycloconverter Drives.*' Proc. 5th Int. Conf. Optimisation of Electric and Electronic Equipments OPTIM'94, Brasov, Romania, Vol. 5, 15-17 May 1996. pp1361-1368.

[5] Costa, A., De Gloria, A., Olivieri, M. '*Hardware Design of Asynchronous Fuzzy Controllers.*' IEEE Trans. Fuzzy Systems, Vol. 4, No. 3, Aug. 1996. pp328-338.

[6] Franklin, G.F., Powell, J.D., Emami-Naeini, A. 'Feedback Control of Dynamic Systems.' Addison-Wesley, Third Edition, 1994.

[7] Hayes, J.P. 'Introduction to Digital Logic Design.' Addison-Wesley, 1993.

[8] Hollstein, T., Halgamuge, S.K., Glesner, M. '*Computer-Aided Design of Fuzzy Systems based on Generic VHDL Specifications.*' IEEE Trans. on Fuzzy Systems, Vol. 4, No. 4, Nov. 1996. pp403-416.

[9] Holtz, J. '*Pulse Width Modulation for Electronic Power Conversion.*' IEEE Proc., Vol. 82, No. 8, Aug. 1994. pp1194-1214.

[10] Horrowits, P., Hill, W. 'The Art of Electronics.' Cambridge University Press, 1995.

[11] Jong, C.C., Ng, L.S., Ho, V.M., Cham, L.P. '*Exploration of Design Architectures in Rapid Prototyping with FPGA.*' Proc. 7th Int. Symposium IC Technology, Systems & Applications ISIC'97, 10-12 Sept. 1997, Singapore. pp426-429.

[12] Kolar, J.W., Thomas, M.W., Schrodl, '*Analytical Calculation of the RMS Current Stress on the D.C. Link Capacitor of Voltage D.C. Link PWM Converter Systems.*' Proc. 9th Int. Conf. on Electrical Machines and Drives, UK, Sep. 1999. pp81-89.

[13] Lander, C.W., 'Power Electronics' McGraw Hill, 1993.

[14] Morley, D., Patterson, E.B. *'Electronic Design Automation (EDA) Approach to Power Electronic Control and Signal Processing.'* Proc. 4th European Conf. Power Electronics and Applications, Florence, Vol. 4, 1991. pp408-412.

[15] Ogata, K. 'Modern Control Engineering.' Prentice Hall, 1990.

[16] Onan RV Genset Installation Manual.

[17] Onan RV Genset Operator's Manual.

[18] Patterson, E.B. *'Electronic Design Automation for Power Electronic Drives.'* PhD Thesis, Nottingham Trent University, 1993.

[19] Semikron Data Book

[20] Tzou, Y., Hsu, H. *'FPGA Realisation of Space-Vector PWM Control IC for Three-Phase PWM Inverter.'* IEEE Trans. Power Electronics, Vol. 12, No. 6, Nov. 1997. pp953-963.

[21] Xilinx Inc., 'The Programmable Logic Data Book.', 1998.

PUBLICATIONS

- *'PWM Control for Variable Speed Stand Alone Generators.'* Proc. of 6th Int. Conf. on Optimisation of E&E Equipments (OPTIM), Vol.II, Brasov, Romania, 14-15 May 1998. pp379-382.

- *'Control of Stand Alone Synchronous Generators at Optimum Speed.'* Proc. of 33rd Intersociety Energy Conversion Engineering Conference (IECEC), Colorado Springs, USA, 2-6 Aug 1998. I-046.

- *'VHDL Design of an Intelligent Fuzzy Logic Controller for Synchronous Generator Sets Implemented in FPGA.'* Proc. of Int. HDL Conference, California, USA, 6-9 April 1999. pp43-47.

- *'An Intelligent Fuzzy Logic Controller for Power Generation Systems.'* Proc. 9th Int. Conf. on Electrical Machines and Drives, Cantebury, UK, Sep. 1999. pp321-324.

Appendix A

VHDL CODE: Simulation

Plant models

App. A-1 : Model of Generator and Rectifier

```
--
-- File:  Genrect.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Model of Synchronous Generator-Rectifier system.
--        Steady state model of generator-rectifier system
--        is derived from the equivalent circuit model
--        of synchronous generators.
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;

entity Genrect is
   port (
            -- inputs
          n        : in REAL;
          Ifield   : in REAL;
          Idc      : in REAL;
          theta    : in REAL;
          CLK      : in STD_LOGIC;
          -- outputs
          Vph      : out REAL;
          Vdc      : out REAL;
          torque   : out REAL);
end Genrect;

architecture Genrect_arc of Genrect is
          constant PI                :REAL:=3.1416;
          constant KG_CONST  :REAL:=4.7997;
          constant R_CONST           :REAL:=0.0015;
          constant X_CONST           :REAL:=1.0000;
          constant RECT1_CONST       :REAL:=0.8165;
          constant RECT2_CONST       :REAL:=1.3500;
          -- Star Configuration
          constant YD_CURR           :REAL:=1.0000;
          constant YD_VOLT           :REAL:=1.7320;
begin
-- REM: Architecture is Synchronous and Sequential

GENREC_PROCESS:
process(CLK)
          variable Ei_VAR                    : REAL;
          variable Iph_VAR, Vph_VAR : REAL;
```

```
        variable torque_VAR             : REAL;
        variable delta_VAR                  : REAL;
begin
   if (CLK'event and CLK='1') then

   -- Avoid division by zero when n=0.0
   if (n=0.0) then
        -- Assign output signals
        -- No speed -> No voltage
        Vph <= 0.0;
        Vdc <= 0.0;
        torque<=50.0;
   else
        Ei_VAR    := KG_CONST * n * Ifield;
        Iph_VAR   := RECT1_CONST * YD_CURR * Idc;
        delta_VAR := arcsin( (Iph_VAR/Ei_VAR) * (X_CONST*cos(theta) + R_CONST*sin(theta)) );
        Vph_VAR := Ei_VAR*cos(delta_VAR) + Iph_VAR*(X_CONST*sin(theta) -
                        R_CONST*cos(theta) );

        -- Assign output signals
        Vph <= Vph_VAR;
        Vdc <= YD_VOLT * RECT2_CONST * Vph_VAR;
        torque<=(6.0*PI*Vph_VAR*Iph_VAR*cos(theta))/n;
   end if;
   end if;
    end process;
end Genrect_arc;

-- Configuration
configuration Genrect_conf1 of Genrect is
      for Genrect_arc
      end for;
end Genrect_conf1;
```

App. A-2 : Model of Diesel Engine

```
--
-- File:  Engine.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Model of Diesel Engine.
--        Based on linearised engine torque-speed characteristics.
--

library ieee;
use ieee.MATH_REAL.all;
use ieee.std_logic_1164.all;

entity Engine is
   port (
        -- inputs
        TL      : in REAL;
        Q       : in REAL;
        Period  : in REAL;
        CLK     : in STD_LOGIC;
        -- output
        Te      : out REAL;
        Nout    : out REAL);
end Engine;
```

```
architecture Engine_arc of Engine is
     signal Nz_SIG       : REAL:=0.0;
     constant A_CONST     :REAL:= 0.9;
     constant B_CONST     :REAL:= 50.53;
     -- Total inertia
     constant J_CONST     :REAL:= 10.0;
begin
-- REM Architecture is Synchronous and Sequential

 ENGINE_PROCESS:
process(CLK)
   variable Te_VAR   :REAL:=0.0;
   variable N_VAR    :REAL:=0.0;
   variable TL_VAR   :REAL:=0.0;
begin
   if (CLK'event and CLK='1') then
        TL_VAR := TL;
        Te_VAR := (B_CONST*Q)-(A_CONST*Nz_SIG);
        N_VAR  := (((Te_VAR-TL_VAR)/J_CONST)*Period)+Nz_SIG;

        -- Assign output/signal with MAX/MIN limit
        if (N_VAR > -5000.0 or N_VAR < 5000.0) then
            Nz_SIG <= N_VAR;
            Nout <= N_VAR;
        elsif (N_VAR < -5000.0) then
            Nz_SIG <= -5000.0;
            Nout <= -5000.0;
       elsif (N_VAR > 5000.0) then
            Nz_SIG <= 5000.0;
            Nout <= 5000.0;
       end if;

       if (Te_VAR > -5000.0 or Te_VAR < 5000.0) then
            Te <= Te_VAR;
       elsif (Te_VAR < -5000.0) then
            Te <= -5000.0;
       elsif (Te_VAR > 5000.0) then
            Te <= 5000.0;
       end if;
   end if;
  end process;
end Engine_arc;


-- Configuration
configuration Engine_conf1 of Engine is
     for Engine_arc
     end for;
end Engine_conf1;
```

Fuzzy Logic Controller

App. A-3 : Input Interface

```
--
-- File:  Interface1.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Part of Fuzzy Logic Controller
--        Simulation Version
--

library ieee;
use ieee.std_logic_1164.all;

entity Interface1 is
   port (
      CLK   :in STD_LOGIC;
      Vdc   :in INTEGER;
      Vref  :in INTEGER;

      x1:out INTEGER;
      x2:out INTEGER);
end Interface1;

architecture Interface1_arc of Interface1 is
begin
    LATCH_PROCESS:
    process(CLK)
        variable NOW_VAR   :INTEGER:=0;
        variable PAST_VAR  :INTEGER:=0;
        variable error     :INTEGER    :=0;
        variable DIFF      :INTEGER    :=0;
begin
if CLK'event and CLK='1' then

    -- error
    error:= (Vdc-Vref)/3;

    -- x1 -> (x1, x2)
    PAST_VAR:=NOW_VAR;
    NOW_VAR:=error;


    -- Output Assignment
    if (NOW_VAR<=(-127)) then
        x1<=(-127);
    elsif (NOW_VAR>=128) then
        x1<=128;
    else
        x1<=NOW_VAR;
    end if;

    DIFF:= NOW_VAR-PAST_VAR;
    if (DIFF>=30) then
        x2<=100;
    elsif (DIFF<=-30) then
```

```
            x2<=-100;
        else
            x2<=DIFF*3;
        end if;
        x2<=0;

 end if;
 end process;
end Interface1_arc;

-- Configuration
configuration Interface1_conf1 of Interface1 is
    for Interface1_arc
    end for;
end Interface1_conf1;
```

App. A-4 : Fuzzifier

```
--
-- File:  Fuzzify.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Fuzzifier
--        Part of Fuzzy Logic Controller
--        Simulation Version
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Fuzzify is
   port (
      -- inputs
      x1: in INTEGER range -127 to 128;
      x2: in INTEGER range -127 to 128;
      -- fuzzy sets for x1
      B1_1: out INTEGER range 0 to 128;
      B1_2: out INTEGER range 0 to 128;
      B1_3: out INTEGER range 0 to 128;
      B1_4: out INTEGER range 0 to 128;
      B1_5: out INTEGER range 0 to 128;
      -- fuzzy sets for x2
      B2_1: out INTEGER range 0 to 128;
      B2_2: out INTEGER range 0 to 128;
      B2_3: out INTEGER range 0 to 128;
      B2_4: out INTEGER range 0 to 128;
      B2_5: out INTEGER range 0 to 128);
end Fuzzify;

architecture Fuzzify_arc of Fuzzify is
   constant a1:INTEGER:=-60;
   constant b1:INTEGER:=-10;
   constant a2:INTEGER:=-60;
   constant b2:INTEGER:=-10;
   constant c2:INTEGER:=0;
   constant a3:INTEGER:=-10;
   constant b3:INTEGER:=0;
   constant c3:INTEGER:=10;
```

```
    constant a4:INTEGER:=0;
    constant b4:INTEGER:=10;
    constant c4:INTEGER:=60;
    constant a5:INTEGER:=10;
    constant b5:INTEGER:=60;
begin
 -- Concurrent Architechture
 ------------------------
 --| Fuzzify input x1 |--
 ------------------------
-- 1. Very Small
B1_1 <= 100 when x1<=a1 else
        (100*(x1-b1))/(a1-b1) when (x1>a1 and x1<=b1) else
        0;
-- 2. Small
B1_2 <= (100*(x1-a2))/(b2-a2) when (x1>=a2 and x1<=b2) else
      (100*(x1-c2))/(b2-c2) when (x1>b2 and x1<=c2) else
      0;
-- 3. Optimum
B1_3 <= (100*(x1-a3))/(b3-a3) when (x1>=a3 and x1<=b3) else
      (100*(x1-c3))/(b3-c3) when (x1>b3 and x1<=c3) else
      0;
-- 4. Big
B1_4 <= (100*(x1-a4))/(b4-a4) when (x1>=a4 and x1<=b4) else
      (100*(x1-c4))/(b4-c4) when (x1>b4 and x1<=c4) else
      0;
-- 5. Very Big
B1_5 <= 0 when x1<=a5 else
      (100*(x1-b5))/(a5-b5)  when (x1>a5 and x1<b5) else
        100;

------------------------
--| Fuzzify input x2 |--
------------------------
-- 1. Very Small
B2_1 <= 100 when x2<=a1 else
        (100*(x2-b1))/(a1-b1) when (x2>a1 and x2<=b1) else
        0;
-- 2. Small
B2_2 <= (100*(x2-a2))/(b2-a2) when (x2>=a2 and x2<=b2) else
      (100*(x2-c2))/(b2-c2) when (x2>b2 and x2<=c2) else
      0;
-- 3. Optimum
B2_3 <= (100*(x2-a3))/(b3-a3) when (x2>=a3 and x2<=b3) else
      (100*(x2-c3))/(b3-c3) when (x2>b3 and x2<=c3) else
      0;
-- 4. Big
B2_4 <= (100*(x2-a4))/(b4-a4) when (x2>=a4 and x2<=b4) else
      (100*(x2-c4))/(b4-c4) when (x2>b4 and x2<=c4) else
      0;
-- 5. Very Big
B2_5 <= 0 when x2<=a5 else
      (100*(x2-b5))/(a5-b5)  when (x2>a5 and x2<b5) else
        100;
end Fuzzify_arc;

-- Configuration
configuration Fuzzify_conf1 of Fuzzify is
    for Fuzzify_arc
    end for;
end Fuzzify_conf1;
```

App. A-5 : Rule Base and Inference Engine

```
--
-- File:  Infer.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Rule base and Inference Engine
--        Part of Fuzzy Logic Controller
--        Simulation Version
--

library ieee;
use ieee.std_logic_1164.all;

entity Infer is
   port (
     CLK: in STD_LOGIC;
     -- Inputs
      B1_1: in INTEGER range 0 to 128;
      B1_2: in INTEGER range 0 to 128;
      B1_3: in INTEGER range 0 to 128;
      B1_4: in INTEGER range 0 to 128;
      B1_5: in INTEGER range 0 to 128;

      B2_1: in INTEGER range 0 to 128;
      B2_2: in INTEGER range 0 to 128;
      B2_3: in INTEGER range 0 to 128;
      B2_4: in INTEGER range 0 to 128;
      B2_5: in INTEGER range 0 to 128;

      -- Outputs
      D1: out INTEGER range 0 to 128;
      D2: out INTEGER range 0 to 128;
      D3: out INTEGER range 0 to 128;
      D4: out INTEGER range 0 to 128;
      D5: out INTEGER range 0 to 128;
      D6: out INTEGER range 0 to 128;
      D7: out INTEGER range 0 to 128;
      D8: out INTEGER range 0 to 128;
      D9: out INTEGER range 0 to 128
      );
end Infer;

architecture Infer_arc of Infer is
begin

Sequential:
process(CLK)
   variable c1,c2,c3,c4,c5: INTEGER range 0 to 128;
   variable c6,c7,c8,c9,c10: INTEGER range 0 to 128;
   variable c11,c12,c13,c14,c15: INTEGER range 0 to 128;
   variable c16,c17,c18,c19,c20: INTEGER range 0 to 128;
   variable c21,c22,c23,c24,c25: INTEGER range 0 to 128;
begin
 -- Fuzzy Inference Engine -- Ci = min(U1_x,U2_y)
 if B1_1 < B2_1 then c1:=B1_1;
 else c1:=B2_1;
 end if;

 if B1_1 < B2_2 then c2:=B1_1;
```

```
 else c2:=B2_2;
 end if;

if B1_1 < B2_3 then c3:=B1_1;
 else c3:=B2_3;
 end if;

if B1_1 < B2_4 then c4:=B1_1;
 else c4:=B2_4;
 end if;

if B1_1 < B2_5 then c5:=B1_1;
 else c5:=B2_5;
 end if;

if B1_2 < B2_1 then c6:=B1_2;
 else c6:=B2_1;
 end if;

if B1_2 < B2_2 then c7:=B1_2;
 else c7:=B2_2;
 end if;

if B1_2 < B2_3 then c8:=B1_2;
 else c8:=B2_3;
 end if;

if B1_2 < B2_4 then c9:=B1_2;
 else c9:=B2_4;
 end if;

if B1_2 < B2_5 then c10:=B1_2;
 else c10:=B2_5;
 end if;

if B1_3 < B2_1 then c11:=B1_3;
 else c11:=B2_1;
 end if;

if B1_3 < B2_2 then c12:=B1_3;
 else c12:=B2_2;
 end if;

if B1_3 < B2_3 then c13:=B1_3;
 else c13:=B2_3;
 end if;

if B1_3 < B2_4 then c14:=B1_3;
 else c14:=B2_4;
 end if;

if B1_3 < B2_5 then c15:=B1_3;
 else c15:=B2_5;
 end if;

if B1_4 < B2_1 then c16:=B1_4;
 else c16:=B2_1;
 end if;

if B1_4 < B2_2 then c17:=B1_4;
 else c17:=B2_2;
 end if;

if B1_4 < B2_3 then c18:=B1_4;
```

```
 else c18:=B2_3;
 end if;

if B1_4 < B2_4 then c19:=B1_4;
 else c19:=B2_4;
 end if;

if B1_4 < B2_5 then c20:=B1_4;
 else c20:=B2_5;
 end if;

if B1_5 < B2_1 then c21:=B1_5;
 else c21:=B2_1;
 end if;

if B1_5 < B2_2 then c22:=B1_5;
 else c22:=B2_2;
 end if;

if B1_5 < B2_3 then c23:=B1_5;
 else c23:=B2_3;
 end if;

if B1_5 < B2_4 then c24:=B1_5;
 else c24:=B2_4;
 end if;

if B1_5 < B2_5 then c25:=B1_5;
 else c25:=B2_5;
 end if;

 ------------------------------------------
 -- 25 Fuzzy Rules -> 9 Fuzzy Sets (Get MAX) --
 ------------------------------------------

 -- Negative Very Big
 D1 <= c25;

 -- Negative Big
 if ( c20=0 and c24=0 ) then    D2<= 0;
 elsif (c20>=c24) then          D2<=c20;
 else                               D2<=c24;
 end if;

 -- Negative
 if (c15=0 and c19=0 and c23=0) then      D3<=0;
 elsif (c15>=c19 and c15>=c23) then  D3<=c15;
 elsif (c19>=c15 and c19>=c23) then  D3<=c19;
 else                                          D3<=c23;
 end if;

 -- Negative Small
 if (c10=0 and c14=0 and c18=0 and c22=0) then      D4<=0;
 elsif (c10>=c14 and c10>=c18 and c10>=c22)then    D4<=c10;
 elsif (c14>=c10 and c14>=c18 and c14>=c22)then    D4<=c14;
 elsif (c18>=c10 and c18>=c14 and c18>=c22) then   D4<=c18;
 else                                                      D4<=c22;
 end if;

 -- Zero
 if (c5 =0 and c9 =0 and c13=0 and
                 c17=0 and c21=0) then            D5<=0;
 elsif (c5>=c9   and c5>=c13 and
                 c5>=c17  and c5>=c21) then  D5<=c5;
```

```
elsif (c9>=c5   and c9>=c13 and
                c9>=c17  and c9>=c21) then  D5<=c9;
elsif (c13>=c5  and c13>=c9 and
                c13>=c17 and c13>=c21)then     D5<=c13;
elsif (c17>=c5  and c17>=c9 and
                c17>=c13 and c17>=c21)then     D5<=c17;
else                                           D5<=c21;
end if;

-- Positive Small
if (c4 =0 and c8 =0 and c12=0 and c16=0) then      D6<=0;
elsif (c4 >=c8  and c4 >=c12 and c4 >=c16) then    D6<=c4;
elsif (c8 >=c4  and c8 >=c12 and c8 >=c12) then    D6<=c8;
elsif (c12>=c4  and c12>=c8  and c12>=c16) then    D6<=c12;
else                                                   D6<=c16;
end if;

-- Positive
if (c3 =0 and c7 =0 and c11=0) then   D7<=0;
elsif (c3 >=c7  and c3 >=c11) then        D7<=c3;
elsif (c7 >=c3  and c7 >=c11) then        D7<=c7;
else                                      D7<=c11;
end if;

 -- Positive Big
 if (c2=0 and c6=0) then    D8<=0;
 elsif (c2>=c6) then            D8<=c2;
 else                           D8<=c6;
 end if;

 -- Positive Very Big
 D9 <= c1;

end process;
end Infer_arc;

-- Configuration
configuration Infer_conf1 of Infer is
    for Infer_arc
    end for;
end Infer_conf1;
```

App. A-6 : Defuzzifier and Output Interface

```
--
-- File:  Defuzz.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Defuzzifier and Output Interface
--        Part of Fuzzy Logic Controller
--        Simulation Version
--

library ieee;
use ieee.std_logic_1164.all;

entity Defuzz is
```

```
port (
  -- Clock
    CLK: in STD_LOGIC;


    D1: in INTEGER range 0 to 128;
    D2: in INTEGER range 0 to 128;
    D3: in INTEGER range 0 to 128;
    D4: in INTEGER range 0 to 128;
    D5: in INTEGER range 0 to 128;
     D6: in INTEGER range 0 to 128;
     D7: in INTEGER range 0 to 128;
     D8: in INTEGER range 0 to 128;
     D9: in INTEGER range 0 to 128;

     -- Crisp control signal
    U: out INTEGER);
end Defuzz;

architecture Defuzz_arc of Defuzz is

    constant E1: INTEGER:=-4;
    constant E2: INTEGER:=-3;
    constant E3: INTEGER:=-2;
    constant E4: INTEGER:=-1;
    constant E5: INTEGER:=0;
    constant E6: INTEGER:=1;
    constant E7: INTEGER:=2;
    constant E8: INTEGER:=3;
    constant E9: INTEGER:=4;

begin

  ------------------
  -- Defuzz Engine --
  ------------------
  -- Weighted average method :

  DEFUZZ_PROCESS:
  process(CLK)
    variable Dividend  :INTEGER:=0;
    variable Divisor       :INTEGER:=1;
    variable Y             :INTEGER;
    variable Uz        :INTEGER:=20;
    variable U_var     :INTEGER;
begin
    if CLK'event and CLK='1' then
        Dividend:=(E1*D1)+(E2*D2)+(E3*D3)+(E4*D4)+(E5*D5)+(E6*D6)+(E7*D7)+(E8*D8)+(E9*D9);
    Divisor:=(D1+D2+D3+D4+D5+D6+D7+D8+D9);

      -- To avoid division by zero
      if Divisor=0 then
        Y:=0;
      else
      -- Definition of crisp output
          Y := (Dividend/Divisor);
          end if;

Ys<=Y;
          --Output interface
          U_var  :=Uz+Y;
          Uz     :=U_var;
          if (U_VAR<=(-254)) then
              U<=(-254);
```

```
        elsif (U_VAR>=255) then
            U<=255;
        else
        U<=U_var;
        end if;


    end if;
  end process;
end Defuzz_arc;

-- Configuration
configuration Defuzz_conf1 of Defuzz is
    for Defuzz_arc
    end for;
end Defuzz_conf1;
```

App. A-7 : Top Hierarchy of FLC

```
--
-- File:  Controller.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Top hierarchy of FLC
--        Binds all the components together
--        Simulation Version
--

library ieee;
use ieee.std_logic_1164.all;

entity Controller is
   port (
      CLK       :in STD_LOGIC;
      Vdc       :in INTEGER;
      Vref      :in INTEGER;
      U         :out INTEGER);
end Controller;

architecture Controller_arc of Controller is
  ---------------------------
  -- Component Declaration --
  ---------------------------
  component Interface1
     port (
         CLK    :in STD_LOGIC;
         Vdc    :in INTEGER;
         Vref   :in INTEGER;
         x1     :out INTEGER;
         x2     :out INTEGER);
  end component Interface1;

  component Fuzzify
     port (
      -- inputs
      x1: in INTEGER range -127 to 128;
      x2: in INTEGER range -127 to 128;
      -- fuzzy sets for x1
```

```
B1_1: out INTEGER range 0 to 128;
B1_2: out INTEGER range 0 to 128;
B1_3: out INTEGER range 0 to 128;
B1_4: out INTEGER range 0 to 128;
B1_5: out INTEGER range 0 to 128;
-- fuzzy sets for x2
B2_1: out INTEGER range 0 to 128;
B2_2: out INTEGER range 0 to 128;
B2_3: out INTEGER range 0 to 128;
B2_4: out INTEGER range 0 to 128;
    B2_5: out INTEGER range 0 to 128);
end component Fuzzify;

component Infer
   port (
        CLK: in STD_LOGIC;
   -- Inputs
    B1_1: in INTEGER range 0 to 128;
    B1_2: in INTEGER range 0 to 128;
    B1_3: in INTEGER range 0 to 128;
    B1_4: in INTEGER range 0 to 128;
    B1_5: in INTEGER range 0 to 128;

    B2_1: in INTEGER range 0 to 128;
    B2_2: in INTEGER range 0 to 128;
B2_3: in INTEGER range 0 to 128;
B2_4: in INTEGER range 0 to 128;
B2_5: in INTEGER range 0 to 128;

-- Outputs
D1: out INTEGER range -127 to 128;
D2: out INTEGER range -127 to 128;
D3: out INTEGER range -127 to 128;
D4: out INTEGER range -127 to 128;
D5: out INTEGER range -127 to 128;
D6: out INTEGER range -127 to 128;
D7: out INTEGER range -127 to 128;
D8: out INTEGER range -127 to 128;
D9: out INTEGER range -127 to 128
);
end component Infer;

component Defuzz
   port (
   -- Clock
    CLK: in STD_LOGIC;
    D1: in INTEGER range -127 to 128;
    D2: in INTEGER range -127 to 128;
    D3: in INTEGER range -127 to 128;
    D4: in INTEGER range -127 to 128;
    D5: in INTEGER range -127 to 128;
    D6: in INTEGER range -127 to 128;
    D7: in INTEGER range -127 to 128;
    D8: in INTEGER range -127 to 128;
    D9: in INTEGER range -127 to 128;

     -- Crisp control signal
     U :out INTEGER);
end component Defuzz;

-----------------------
-- Signal Declaration --
-----------------------
signal x1_SIG, x2_SIG : INTEGER;
```

```
 signal SB1_1,SB1_2,SB1_3,SB1_4,SB1_5 : INTEGER range 0 to 128;
 signal SB2_1,SB2_2,SB2_3,SB2_4,SB2_5 : INTEGER range 0 to 128;
 signal D1_SIG,D2_SIG,D3_SIG : INTEGER range -127 to 128;
 signal D4_SIG,D5_SIG,D6_SIG : INTEGER range -127 to 128;
 signal D7_SIG,D8_SIG,D9_SIG : INTEGER range -127 to 128;

begin
  ----------------------------------------
  -- Component Instantiation (Port Map) --
  ----------------------------------------
Interface1_U: Interface1
   port map(
            -- Inputs
            CLK=>CLK,
            Vdc=>Vdc,
            Vref=>Vref,
            -- Outputs
            x1=>x1_SIG,
            x2=>x2_SIG
            );

Fuzzify_U: Fuzzify
   port map(
            -- Inputs
            x1=>x1_SIG, x2=>x2_SIG,

            -- Outputs
            B1_1=>SB1_1,
            B1_2=>SB1_2,
            B1_3=>SB1_3,
            B1_4=>SB1_4,
            B1_5=>SB1_5,

            B2_1=>SB2_1,
            B2_2=>SB2_2,
            B2_3=>SB2_3,
            B2_4=>SB2_4,
            B2_5=>SB2_5
            );

Infer_U: Infer
   port map(
            CLK=>CLK,
            -- Inputs
            B1_1=>SB1_1,
            B1_2=>SB1_2,
            B1_3=>SB1_3,
            B1_4=>SB1_4,
            B1_5=>SB1_5,

            B2_1=>SB2_1,
            B2_2=>SB2_2,
            B2_3=>SB2_3,
            B2_4=>SB2_4,
            B2_5=>SB2_5,

            -- Outputs
            D1=>D1_SIG,D2=>D2_SIG,D3=>D3_SIG,
            D4=>D4_SIG,D5=>D5_SIG,D6=>D6_SIG,
            D7=>D7_SIG,D8=>D8_SIG,D9=>D9_SIG
            );

 Defuzz_U:Defuzz
```

```
port map(
        -- Inputs
        CLK=>CLK,
        D1=>D1_SIG,D2=>D2_SIG,D3=>D3_SIG,
        D4=>D4_SIG,D5=>D5_SIG,D6=>D6_SIG,
        D7=>D7_SIG,D8=>D8_SIG,D9=>D9_SIG,

        -- Output
        U=>u
        );

end Controller_arc;

-- Configuration
configuration Controller_conf1 of Controller is
    for Controller_arc
        for Interface1_U: Interface1 use configuration work.Interface1_conf1;
        end for;

        for Fuzzify_U:Fuzzify use configuration work.Fuzzify_conf1;
        end for;

        for Infer_U:Infer use configuration work.Infer_conf1;
        end for;

        for Defuzz_U:Defuzz use configuration work.Defuzz_conf1;
        end for;
    end for;
end Controller_conf1;
```

Testbenches and Simulations

App. A-8 : Delay

```
--
-- File:  Delay.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Part of Simulation component
--

library ieee;
use ieee.std_logic_1164.all;

entity delay is
   port (
      Tin: in REAL;
      Uin: in REAL;
      MUX: in STD_LOGIC;
      Uout: out REAL:=2.0;
      Tout: out REAL:=10.0);
end delay;

architecture delay_arc of delay is
    begin
        Tout <= Tin when MUX='1';
```

```
        Uout <= Uin when MUX='1';
end delay_arc;

-- Configuration
configuration delay_conf1 of delay is
    for delay_arc
    end for;
end delay_conf1;
```

App. A-9 : Simulator

```
--
-- File:  Sim.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Simulation component
--        This test unit comprises:
--        FLC, Engine, Genrect and a delay component
--

library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;


entity Sim is
port (
    -- inputs
    CLK1        : in STD_LOGIC;
    CLK2        : in STD_LOGIC;
    Period      : in REAL;
    Idc         : in REAL;
    theta       : in REAL;
    Ifield      : in REAL;
     Vref       : in INTEGER;
    -- outputs
    TE          : out REAL;
    Vph         : out REAL;
    Vdc         : out REAL);
end Sim;

architecture Sim_arc of Sim is
 ---------------------------
 -- Component Declaration --
 ---------------------------
    component Controller
    port(
        CLK         :in std_logic;
        Vdc         :in INTEGER;
        Vref        :in INTEGER;
        U           :out INTEGER);
    end component;

    component Engine
    port(
     TL         : in REAL;
     U          : in REAL;
     CLK        : in STD_LOGIC;
     Period: in REAL;
```

```
 TE          : out REAL;
 N           : out REAL);
end component;

component Genrect
port(
 N           : in REAL;
 Ifield      : in REAL;
 Idc         : in REAL;
 theta       : in REAL;
   CLK       : in STD_LOGIC;
   Vph       : out REAL;
   Vdc       : out REAL;
   TG        : out REAL);
end component;

component delay
port(
  Tin      : in REAL;
  Uin      : in REAL;
  MUX      : in STD_LOGIC;
  Uout     : out REAL;
  Tout     : out REAL);
end component;

-----------------------
-- Signal Declaration --
-----------------------
 -- Controller input flow
 signal    Vdc_SIG          :REAL;
 signal    Vdc_rSIG         :REAL;
 signal    Vdc_iSIG         :INTEGER;

 -- Controller output flow
 signal    U_iSIG           :INTEGER;
 signal    U_rSIG, Uo_SIG       :REAL;

 -- Torque & Speed signals
 signal TL_SIG   :REAL :=50.0;
 signal TG_SIG :REAL;
   signal N_SIG     :REAL;

    -- Delay element
   signal MUX       :STD_LOGIC  :='0';
   signal UE_SIG    :REAL           :=2.0;
begin
 ---------------------------------------
 -- Component Instantiation (Port Map) --
 ---------------------------------------
    Controller_U: Controller
    port map(
            -- Inputs
            CLK         =>CLK1,
            Vdc         =>Vdc_iSIG,
            Vref        =>Vref,
            -- Outputs
            U           =>U_iSIG);

Engine_U: Engine
port map(
            -- Inputs
            TL          =>TL_SIG,
            U           =>UE_SIG,
            CLK         =>CLK2,
```

```
        Period      =>Period,
        -- Outputs
        TE          =>TE,
        N           =>N_SIG);

Genrect_U: Genrect
port map(
        -- Inputs
        N           =>N_SIG,
        Ifield      =>Ifield,
        Idc         =>Idc,
        theta       =>theta,
        CLK         =>CLK2,
        -- Outputs
        Vph         =>Vph,
        Vdc         =>Vdc_SIG,
        TG          =>TG_SIG);

delay_U: delay
port map(
        Tin         =>TG_SIG,
        Uin     =>Uo_SIG,
        MUX         =>MUX,
        Uout        =>UE_SIG,
        Tout        =>TL_SIG);

-- end of component instantiation

-- Torque in/out delay to overcome problem
-- caused by propagation of unknown values
-- during the starting transient.
MUX <='1' after 30ns;

-- error: Normalise
Vdc_rSIG <= Vdc_SIG * 1.0;

-- error: Real-Integer Conversion
Vdc_iSIG <=   2000 when (Vdc_rSIG>2000.0) else
              0 when (Vdc_rSIG<0.0) else
              INTEGER(Vdc_rSIG);

-- U: Integer-Real Conversion
U_rSIG <= REAL(U_iSIG);

-- UnNormalise U (from -127/128 to 12.7/12.8)
OutputU:
process(CLK1)
    variable COUNT_VAR : STD_LOGIC := '0';
    variable U_VAR    :REAL;
begin
if (CLK1'event and CLK1='1') then
    if (COUNT_VAR='1') then
        U_VAR := (U_rSIG/10.0);

        if (U_VAR>25.5) then
            Uo_SIG <= 25.5;
        elsif (U_VAR<-25.4) then
            Uo_SIG <= -25.4;
        else
            Uo_SIG <= U_VAR;
        end if;
    else
        -- Initial condition
        COUNT_VAR:='1';
```

```
            Uo_SIG <= 2.0;
        end if;
    end if;
    end process;

    --- Assign output Vdc
    Vdc <= Vdc_SIG;

end Sim_arc;
```

```
configuration Sim_conf1 of Sim is
    for Sim_arc
        for Controller_U: Controller use configuration work.Controller_conf1;
        end for;

        for Engine_U: Engine use configuration work.Engine_conf1;
        end for;

        for Genrect_U: Genrect use configuration work.Genrect_conf1;
        end for;

        for delay_U: delay use configuration work.delay_conf1;
        end for;
    end for;
end Sim_conf1;
```

App. A-10 : Testbench for 'Sim'

```
--
-- File:  TB_Sim.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Testbench for Sim
--        Provides the appropriate stimuli and observes the simulated signals
--        Writes the observed values into a text file
--
```

```
library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;
use std.textio.all;

entity TB_Sim is
end TB_Sim;

architecture TB_Sim_arc of TB_Sim is
    -- Component declaration of the tested unit
    component Sim
    port(
        --Inputs
        CLK1   :in std_logic;
        CLK2   :in std_logic;
        Period  in REAL;
        Idc :in REAL;
        theta    :in REAL;
        Ifield   :in REAL;
        Vref     :in INTEGER;
        --Outputs
        TE :out REAL;
```

```
    Vph     :out REAL;
    Vdc     :out REAL );
end component;

-- Stimulus signals - signals mapped to the input and inout ports of tested entity
signal CLK1    : std_logic:='1';
signal CLK2    : std_logic:='1';
signal Period  : REAL;
signal Idc     : REAL;
signal theta   : REAL;
    signal Ifield   : REAL;
    signal Vref     : INTEGER;

    -- Observed signals - signals mapped to the output ports of tested entity
    signal TE       : REAL;
    signal Vph : REAL;
    signal Vdc : REAL;

    -- Signals for simulation purposes

begin

    -- Unit Under Test port map
    UUT : Sim
        port map(
            --Inputs
            CLK1  => CLK1,
            CLK2  => CLK2,
            Period => Period,
            Idc => Idc,
            theta   => theta,
            Ifield  => Ifield,
            Vref    => Vref,
            --Outputs
            TE      => TE,
            Vph     => Vph,
            Vdc     => Vdc );

-------------------------
-- ***Stimulus***--
-------------------------
    CLK1 <= not CLK1 after 10ns;
    CLK2 <= not CLK2 after 1ns;
    -- Period:
    Period  <= 3.0;
    theta       <= 0.0;
    Ifield      <= 2.5;
    Vref        <= 1000;
    Idc     <= 25.0,
               1.0 after 4100ns;

--Write results into file
process (CLK1)
    file outfile : text is out
    "C:\My Designs\Simulation\src\Results\Further25.txt";
    variable out_line    : line;
begin
    write(out_line, Vdc);
--  write(out_line, "   "); write(out_line, Idc);
    writeline(outfile, out_line);
end process;

end TB_Sim_arc;
```

```
configuration TB_Sim_conf1 of TB_Sim is
    for TB_Sim_arc
        for UUT : Sim
            use entity work.Sim(Sim_ARC);
        end for;
    end for;
end TB_Sim_conf1;
```

Appendix B

VHDL CODE: Synthesis & Implementation

Fuzzy Logic Controller

App. B-1 : Input Interface

```
--
-- File:  Deriv.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Part of Fuzzy Logic Controller
--        Implementation Version
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity Deriv is
   port (
      CLK       : in STD_LOGIC;
      RST       : in STD_LOGIC;
      Vdc       : in std_logic_vector(7 downto 0);
      Vref      : in std_logic_vector(7 downto 0);

      x1: out std_logic_vector(8 downto 0);
       x2: out std_logic_vector(8 downto 0));
end Deriv;

architecture Deriv_arc of Deriv is
begin
LATCH_PROCESS:
process(CLK,RST)
     variable x: std_logic_vector(8 downto 0);
     variable NOW_VAR: std_logic_vector(8 downto 0);
     variable PAST_VAR: std_logic_vector(8 downto 0);
     variable DIFF: std_logic_vector(8 downto 0);
     variable Vdc_var: std_logic_vector(8 downto 0);
     variable Vref_var: std_logic_vector(8 downto 0);
     variable x_temp: std_logic_vector(8 downto 0);
     variable error: std_logic_vector(8 downto 0);
 begin --process
 if RST='1' then
      NOW_VAR :="000000000";
      PAST_VAR :="000000000";
      DIFF:= "000000000";
      x1<= "000000000";
      x2<= "000000000";
elsif CLK'event and CLK='1' then
```

```
--Convert from unsigned to signed
Vdc_var(8):='0';
Vdc_var(7 downto 0):=Vdc;
Vref_var(8):='0';
Vref_var(7 downto 0):=Vref;

--Get Error
error:=Vdc_var-Vref_var;
--x is error*(Gain=3)
x:=error+shl(error,"1");

--Overflow check
if (error(8) XOR x(8))='1' then
     if error(8)='1' then
          x:="110011100";
     else
          x:="001100100";
     end if;
end if;

----------------------
--Block: x -> x1, x2--
----------------------
     -- Effect immediately
          PAST_VAR:=NOW_VAR;
          NOW_VAR:=x;
          DIFF:= NOW_VAR-PAST_VAR;

          --x1 is NOW_VAR
          x1<=NOW_VAR;

          ------
          --x2--
          ------
          x2<="000000000";
                    -- Amplifier : Gain 3, Output Saturate 90
--        if (conv_integer(DIFF)>=30) then
               --x2=90
--             x2<="001011010";
--        elsif (conv_integer(DIFF)<=-30) then
               --x2=(-90)
--             x2<="110100110";
--        else
               --Multiply by 3
--             x2<=DIFF+shl(DIFF,"1");
--   end if; -- Amplifier

end if; -- Clock,Reset

end process;
end Deriv_arc;

-- Configuration
configuration Deriv_conf1 of Deriv is
     for Deriv_arc
     end for;
end Deriv_conf1;
library ieee;
use ieee.std_logic_1164.all;
```

App. B-2 : Fuzzifier

```
--
-- File:  Fuzzify2.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Part of Fuzzy Logic Controller
--        Implementation Version
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity Fuzzify2 is
   port (
      -- inputs
      CLK        : in STD_LOGIC;
      RST        : in STD_LOGIC;
      x1      : in std_logic_vector(8 downto 0);
      x2      : in std_logic_vector(8 downto 0);

      -- address
      ADR1: out std_logic_vector(1 downto 0);
      ADR2: out std_logic_vector(1 downto 0);

      -- fuzzy sets for x1 (data)
      B1_A : out std_logic_vector(8 downto 0);
      B1_B : out std_logic_vector(8 downto 0);

      -- fuzzy sets for x2 (data)
      B2_A : out std_logic_vector(8 downto 0);
      B2_B : out std_logic_vector(8 downto 0);

      READY   : out std_logic);
end Fuzzify2;

architecture Fuzzify2_arc of Fuzzify2 is
      signal temp      : std_logic_vector(1 downto 0);
      signal temp_A  : std_logic_vector(8 downto 0);
      signal temp_B  : std_logic_vector(8 downto 0);
      signal R_sig          : std_logic;
begin

-- Sequential Architecture (Synchronous)
process(CLK,RST)
      constant AA_const:std_logic_vector(8 downto 0):=conv_std_logic_vector(-60,9);
      constant BB_const:std_logic_vector(8 downto 0):=conv_std_logic_vector(-10,9);
      constant CC_const:std_logic_vector(8 downto 0):="000000000";
      constant DD_const:std_logic_vector(8 downto 0):=conv_std_logic_vector(10,9);
      constant EE_const:std_logic_vector(8 downto 0):=conv_std_logic_vector(60,9);

      constant hundred: std_logic_vector(8 downto 0):="001100100";
      constant zero: std_logic_vector(8 downto 0):="000000000";

      variable dumbs: std_logic_vector(8 downto 0);

      variable x: std_logic_vector(8 downto 0);
      variable ADR: std_logic_vector(1 downto 0);
      variable B_A: std_logic_vector(8 downto 0);
      variable B_B: std_logic_vector(8 downto 0);
```

```
begin
if RST='1' then
        ADR1<="00";
        ADR2<="00";

        -- fuzzy sets for x1 (data)
        B1_A<= zero;
        B1_B<= zero;
        -- fuzzy sets for x2 (data)
        B2_A<= zero;
        B2_B<= zero;

        R_sig<='1';
        READY<='0';
elsif CLK'event and CLK='1' then

--MUX
if R_sig='1' then
     x:=x1;
else
     x:=x2;
end if; --select bit

  ------------------------------------
--| Fuzzify input x    (Sequential) |--
------------------------------------

    if x<=AA_const then
        --Zone 0a
        ADR:="00";
        --Very Small(B_1)
        B_A:=hundred;
        --Small(B_2)
        B_B:=zero;

    elsif (x>AA_const and x<=BB_const) then
        --Zone 0b
        ADR:="00";
        --Very Small(B_1)=2(-x-10)
        dumbs:=not(x-"01")-"01010";
        B_A :=shl(dumbs,"1");
        --Small(B_2)=100-2(-x-10)
        B_B:=hundred-B_A;

   elsif (x>BB_const and x<=CC_const) then
        --Zone1
        ADR:="01";
        --Small(B_2)=x*(-10)=pos_x*10
        dumbs:=not(x-"01");
        B_A:=shl(dumbs,"1")+shl(dumbs,"11");
        --Optimum(B_3)
        B_B:=hundred-B_A;

    elsif (x>CC_const and x<=DD_const) then
        --Zone2
        ADR:="10";
        --Optimum(B_3)
        dumbs:=shl(x,"1")+shl(x,"11");
        B_A:=hundred-dumbs;
        --Big(B_4)
        B_B:=dumbs;

    elsif (x>DD_const and x<=EE_const) then
```

```
    --Zone3a
    ADR:="11";
    --Big(B_4)=100-2(x-10)
    dumbs:=x-"01010";
    --Very Big(B_5)
    B_B:=shl(dumbs,"1");
    B_A:=hundred-B_B;

else --(x>EE_const)
    --Zone3b
        ADR:="11";
        --Big(B_4)
        B_A:=zero;
        --Very Big(B_5)
        B_B:=hundred;

    end if; -- fuzzy sets for x

  --Storing of process results
if R_sig='1' then
    temp<=ADR;
    temp_A<=B_A;
    temp_B<=B_B;
    READY<='0';
    R_sig<='0';
elsif R_sig='0' then
    ADR1<=temp;
    B1_A<=temp_A;
    B1_B<=temp_B;

    ADR2<=ADR;
    B2_A<=B_A;
    B2_B<=B_B;

    READY<='1';
    R_sig<='1';
end if; --R_sig
end if; --CLK,RESET
end process; -- main

end Fuzzify2_arc;

-- Configuration
configuration Fuzzify2_conf1 of Fuzzify2 is
    for Fuzzify2_arc
    end for;
end Fuzzify2_conf1;
```

App. B-3 : Inference Engine

```
--
-- File:  Infer.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Part of Fuzzy Logic Controller
--        Implementation Version
--

library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity Infer is
   port (

      -- Standard Input
      CLK    : in std_logic;
      LOAD: in std_logic;
      RST    : in std_logic;

      -- Inputs
      ADR1: in std_logic_vector(1 downto 0);
      B1_Ad: in std_logic_vector(8 downto 0);
      B1_Bd: in std_logic_vector(8 downto 0);

      ADR2: in std_logic_vector(1 downto 0);
      B2_Ad: in std_logic_vector(8 downto 0);
      B2_Bd: in std_logic_vector(8 downto 0);

      -- Outputs
      win: out std_logic_vector(3 downto 0);

      c1: out std_logic_vector(8 downto 0);
      c2: out std_logic_vector(8 downto 0);
      c3: out std_logic_vector(8 downto 0);
      c4: out std_logic_vector(8 downto 0));
end Infer;

architecture Infer_arc of Infer is
begin

MAIN_PROCESS:
process(CLK,RST)
begin --process

if RST='1' then
      win<="0000";
      c1<="000000000";
      c2<="000000000";
      c3<="000000000";
      c4<="000000000";
elsif CLK'event and CLK='1' then
      if LOAD='1' then
-----------------------------------------------
--| Fuzzy Inference Engine | Ci=min(B1_a,B2_b)|--
-----------------------------------------------

win<=("00"&ADR1)+ADR2;

---------------------------------
--| Mini Fuzzy Inference Engine |--
---------------------------------
--:B-->C (min operation)
-- c1:=min(B1_Ad,B2_Ad)
      if (B1_Ad<B2_Ad) then
            c1<=B1_Ad;
      else
            c1<=B2_Ad;
      end if;
-- c2:=min(B1_Bd,B2_Ad)
      if (B1_Bd<B2_Ad) then
            c2<=B1_Bd;
      else
```

```
            c2<=B2_Ad;
        end if;
    -- c3:=min(B1_Ad,B2_Bd)
        if (B1_Ad<B2_Bd) then
            c3<=B1_Ad;
        else
            c3<=B2_Bd;
        end if;
    -- c4:=min(B1_Bd,B2_Bd)
        if (B1_Bd<B2_Bd) then
            c4<=B1_Bd;
        else
            c4<=B2_Bd;
        end if;

        end if; --LOAD
    end if; --RST,CLK

    end process;
    end Infer_arc;

    -- Configuration
    configuration Infer_conf1 of Infer is
        for Infer_arc
        end for;
    end Infer_conf1;
```

App. B-4 : Defuzzifier

```
--
-- File:  Defuzz.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--          Part of Fuzzy Logic Controller
--          Implementation Version
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity Defuzz is
   port (

      -- Standard Input
      CLK: in std_logic;
      RST: in std_logic;
               LOAD: in std_logic;

      WIN: in std_logic_vector(3 downto 0);

      c1: in std_logic_vector(8 downto 0);
      c2: in std_logic_vector(8 downto 0);
      c3: in std_logic_vector(8 downto 0);
      c4: in std_logic_vector(8 downto 0);

      -- Outputs
               READY: out std_logic;
```

```
            divA: out std_logic_vector(13 downto 0);
            divB: out std_logic_vector(8 downto 0));
end Defuzz;

architecture Defuzz_arc of Defuzz is
    signal READY_sig: std_logic;
begin

MAIN_PROCESS:
process(CLK,RST)

variable COUNT: std_logic_vector(3 downto 0);

variable DA: std_logic_vector(8 downto 0);
variable DB: std_logic_vector(8 downto 0);
variable DC: std_logic_vector(8 downto 0);

variable VA1: std_logic_vector(13 downto 0);
variable VB1: std_logic_vector(13 downto 0);
variable VC1: std_logic_vector(13 downto 0);

variable VA: std_logic_vector(13 downto 0);
variable VB: std_logic_vector(13 downto 0);
variable VC: std_logic_vector(13 downto 0);

    variable SA: std_logic_vector(13 downto 0);
    variable SB: std_logic_vector(13 downto 0);
    variable SC: std_logic_vector(13 downto 0);

begin --process

if RST='1' then
    divA<="00000000000000";
    divB<="000000001";
    READY<='1';
    READY_sig<='1';
    COUNT:="0000";
    DA:="000000000";
    DB:="000000000";
    DC:="000000000";
    VA:="00000000000000";
    VB:="00000000000000";
    VC:="00000000000000";
elsif CLK'event and CLK='1' then

 if LOAD='1' and (READY_sig='1' or (COUNT>WIN)) then
    COUNT:="0000";
    --Sample
    --DA:=c1
    DA:=c1;
    -- DB:=max(c2,c3)
    if c2>c3 then DB:=c2;
    else DB:=c3;
    end if;
    --DC:=c3
    DC:=c4;

    --Defuzz: Multiplication
    --VA:=DA*POS_40;
     VA1:="00000"&DA;
--       SA:=shl(VA1,"1")+shl(VA1,"11"); --VA1*10
     SA:=VA1;
     VA:=shl(SA,"10"); --SAv*4
    --VB:=DB*POS_30;
```

```
        VB1:="00000"&DB;
--          SB:=shl(VB1,"1")+shl(VB1,"11"); --VB1*10
        SB:=VB1;
        VB:=shl(SB,"1")+SB; --SAc*3
      --VC:=DC*POS_20;
        VC1:="00000"&DC;
--          SC:=shl(VC1,"1")+shl(VC1,"11");  --VC1*10
        SC:=VC1;
        VC:=shl(SC,"1"); --SCv*2
elsif READY_sig='0' then
        COUNT:=COUNT+1;
        VA:=VA-SA;
        VB:=VB-SB;
        VC:=VC-SC;
else
        --LOAD=0, READY_sig=1
        --Do nothing
end if;  --READY,LOAD

if COUNT=WIN then
        --Inputs to the divider
        divA<=VA+VB+VC;
        divB<=DA+DB+DC;
        READY_sig<='1';
        READY<='1';
else
        READY_sig<='0';
        READY<='0';
end if; --WIN=COUNT

end if; --RST,CLK
end process;
end Defuzz_arc;

-- Configuration
configuration Defuzz_conf1 of Defuzz is
        for Defuzz_arc
        end for;
end Defuzz_conf1;
```

App. B-5 : Divider

```
--
-- File:  Divide.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--          Part of Fuzzy Logic Controller
--          Implementation Version
--          Includes output interface
--         Output of divider is Y
--         Output of interface is U
--          Overflow limit included
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
```

```
entity Divider is
   port (

     -- Standard Input
     CLK: in std_logic;
     RST: in std_logic;
     LOAD: in std_logic;

     -- Inputs
      divA: in std_logic_vector(13 downto 0);
      divB: in std_logic_vector(8 downto 0);

       -- Outputs
     READY: out std_logic;
     U: out std_logic_vector(7 downto 0));
end Divider;

architecture Divider_arc of Divider is
     signal READY_sig: std_logic;
begin

MAIN_PROCESS:
process(CLK,RST)
     --Variables for divider
     variable SIGN: std_logic;
     variable divBp: std_logic_vector(13 downto 0);
     variable divBn: std_logic_vector(13 downto 0);
     variable A: std_logic_vector(13 downto 0);
     variable divA_var: std_logic_vector(12 downto 0);
     variable Y1_var: std_logic_vector(12 downto 0);
     variable Y: std_logic_vector(7 downto 0);
     variable U_past: std_logic_vector(7 downto 0);
     variable U_var: std_logic_vector(7 downto 0);
     variable COUNT: integer range -1 to 11;

begin
if RST='1' then
     U<="01111111";
     U_past:="01111111";
     READY_sig<='1';
     READY<='0';
     COUNT:=11;
elsif CLK'event and CLK='1' then

--Loading new input values and perform first division sequence
--Condition: Load=1 & Ready=1
if LOAD='1' and READY_sig='1' then

--Division by zero check
if divB="00000000" then
     --Avoid division by zero: assign Y=0
     Y1_var:="0000000000000";
     READY<='1';
     READY_sig<='1';
else --divB
     --Assign D(+ve) and D(-ve)
     divBp:="00000"&divB;
     divBn:=not(divBp)+"0000000000001";

     --Convert SIGNED into UNSIGNED
     SIGN:=divA(13);
     if SIGN='1' then
          divA_var:=not(divA(12 downto 0))+"01";
     else
```

```
    divA_var:=divA(12 downto 0);
end if; --SIGN type conversion

--First division sequence after loading
A(13 downto 1):="0000000000000";
A(0):=divA_var(12);
divA_var:=shl(divA_var,"1");
A:=A+divBn;
Y1_var(12):=not(A(13));
COUNT:=11; --END First division sequence

end if;  --divB

--Subsequent Division sequence.
--Condition: Load=[don't care] & Ready=0
elsif (READY_sig='0' and COUNT>0) then
-----------------------------------------
-- DIVIDER                             --
-- divA: in std_logic_vector(13 downto 0);  --
-- divB: in std_logic_vector(7 downto 0);   --
-- Y: out std_logic_vector(7 downto 0);     --
-- U: out std_logic_vector(7 downto 0);     --
-----------------------------------------

        COUNT:=COUNT-1;
        A:=shl(A,"1");
        A(0):=divA_var(12);
        divA_var:=shl(divA_var,"1");

        if Y1_var(COUNT+1)='0' then
            A:=A+divBp;
        else
            A:=A+divBn;
        end if;
        Y1_var(COUNT):=not(A(13));

--LOAD=0, READY=1
--No operation

end if; --LOAD, READY

    if COUNT=0 then
        --Ready to spit out the answer
        --Converts back into SIGNED value (2's complement)
        --Assume that Y2_var's value does not exceed 8bits(signed)
        Y:=Y1_var(7 downto 0);
        if SIGN='1' then
--          Y2_var:=not(('0'&Y1_var)-"01");
--          Y<=Y2_var(7 downto 0);
            --Negative
            U_var:=U_Past-Y;
            --Set lower limit "00000000"
            if U_var>U_past then
                U_past:="00000000";
                U<="00000000";
            else
                U_past:=U_var;
                U<=U_var;
            end if;
        else
            --Positive
            U_var:=U_Past+Y;
            --Set upper limit "11111111"
```

```
    if U_var<U_past then
        U_past:="11111111";
        U<="11111111";
    else
        U_past:=U_var;
        U<=U_var;
    end if;
end if; --SIGN

READY<='1';
    READY_sig<='1';

else --COUNT

    --Not ready: condition: COUNT != 0
    READY<='0';
    READY_sig<='0';

end if; --COUNT

end if; --CLK,RST
end process;
end Divider_arc;

-- Configuration
configuration Divider_conf1 of Divider is
    for Divider_arc
    end for;
end Divider_conf1;
```

App. B-6 : Top Hierarchy component of FLC

```
--
-- File:  Control.vhd
-- 1998
-- J.G. Khor
-- Remarks:
--        Fuzzy Logic Controller (Top hierarchy component)
--        Implementation Version
--        Contains Deriv, Fuzzify2, Infer, Defuzz, Divider
--        Remarks: modified to fit 9bits
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity Control is
   port (
      -- inputs
      CLK1: in std_logic;
      CLK2: in std_logic;
      RST: in std_logic;

      Vdc: in std_logic_vector(7 downto 0);
      Vref: in std_logic_vector(7 downto 0);

      --Test Probes
      --x1: out std_logic_vector(8 downto 0);
      --x2: out std_logic_vector(8 downto 0);
```

```
        -- Outputs
        U: out std_logic_vector(7 downto 0);
        READY: out std_logic);
end Control;

architecture Control_arc of Control is

component Deriv
      port(
        CLK: in STD_LOGIC;
        RST: in STD_LOGIC;
        Vdc: in std_logic_vector(7 downto 0);
        Vref: in std_logic_vector(7 downto 0);

        x1: out std_logic_vector(8 downto 0);
        x2: out std_logic_vector(8 downto 0));
end component;

component Fuzzify2
      port(
      -- inputs
        CLK: in STD_LOGIC;
        RST: in STD_LOGIC;
        x1: in std_logic_vector(8 downto 0);
        x2: in std_logic_vector(8 downto 0);

        -- address
        ADR1: out std_logic_vector(1 downto 0);
        ADR2: out std_logic_vector(1 downto 0);

        -- fuzzy sets for x1 (data)
        B1_A: out std_logic_vector(8 downto 0);
        B1_B: out std_logic_vector(8 downto 0);

        -- fuzzy sets for x2 (data)
        B2_A: out std_logic_vector(8 downto 0);
        B2_B: out std_logic_vector(8 downto 0);
        READY: out std_logic);
end component;

component Infer
      port(
      -- Standard Input
              CLK: in std_logic;
      LOAD: in std_logic;
      RST: in std_logic;

      -- Inputs
       ADR1: in std_logic_vector(1 downto 0);
       B1_Ad: in std_logic_vector(8 downto 0);
       B1_Bd: in std_logic_vector(8 downto 0);

       ADR2: in std_logic_vector(1 downto 0);
       B2_Ad: in std_logic_vector(8 downto 0);
       B2_Bd: in std_logic_vector(8 downto 0);

        -- Outputs
        win: out std_logic_vector(3 downto 0);

        c1: out std_logic_vector(8 downto 0);
                c2: out std_logic_vector(8 downto 0);
                c3: out std_logic_vector(8 downto 0);
                c4: out std_logic_vector(8 downto 0));
end component;
```

```
component Defuzz
   port (
     -- Standard Input
     CLK: in std_logic;
     RST: in std_logic;
              LOAD: in std_logic;

     WIN: in std_logic_vector(3 downto 0);
     c1: in std_logic_vector(8 downto 0);
     c2: in std_logic_vector(8 downto 0);
     c3: in std_logic_vector(8 downto 0);
     c4: in std_logic_vector(8 downto 0);

      -- Outputs
     READY: out std_logic;
     divA: out std_logic_vector(13 downto 0);
     divB: out std_logic_vector(8 downto 0));
end component;

component Divider
   port (
        -- Standard Input
     CLK: in std_logic;
     RST: in std_logic;
     LOAD: in std_logic;

     -- Inputs
      divA: in std_logic_vector(13 downto 0);
      divB: in std_logic_vector(8 downto 0);

      -- Outputs
                READY: out std_logic;
                U: out std_logic_vector(7 downto 0));
end component;


--Signal Declaration
signal SIG2_4, SIG4_5: std_logic;
signal x1_sig, x2_sig: std_logic_vector(8 downto 0);
signal adr1_sig, adr2_sig: std_logic_vector(1 downto 0);
signal B1a_sig, B1b_sig, B2a_sig, B2b_sig: std_logic_vector(8 downto 0);
signal c1_sig, c2_sig, c3_sig, c4_sig: std_logic_vector(8 downto 0);
signal win: std_logic_vector(3 downto 0);
signal DA_sig: std_logic_vector(13 downto 0);
signal DB_sig: std_logic_vector(8 downto 0);
signal READY2, READY4: std_logic;

begin

Deriv_U: Deriv
     port map(
     --in
     CLK=>CLK1,
     RST=>RST,
     Vdc=>Vdc,
     Vref=>Vref,
     --out
     x1=>x1_sig,
     x2=>x2_sig);

Fuzzify2_U: Fuzzify2
     port map(
```

```
--in
CLK=>CLK2,
 RST=>RST,
 x1=>x1_sig,
 x2=>x2_sig,
 --out
 ADR1=>adr1_sig,
 ADR2=>adr2_sig,
 B1_A=>B1a_sig,
 B1_B=>B1b_sig,
 B2_A=>B2a_sig,
 B2_B=>B2b_sig,
 READY=>READY2);

Infer_U: Infer
    port map(
    --in
    CLK=>CLK2,
    LOAD=>READY2,
    RST=>RST,
    ADR1=>adr1_sig,
     B1_Ad=>B1a_sig,
     B1_Bd=>B1b_sig,
     ADR2=>adr2_sig,
     B2_Ad=>B2a_sig,
     B2_Bd=>B2b_sig,
     -- Outputs
     win=>win,
     c1=>c1_sig,
    c2=>c2_sig,
    c3=>c3_sig,
    c4=>c4_sig);

Defuzz_U: Defuzz
    port map(
    -- Standard Input
    CLK=>CLK2,
    RST=>RST,
            LOAD=>SIG2_4,
    WIN=>win,
    c1=>c1_sig,
    c2=>c2_sig,
    c3=>c3_sig,
    c4=>c4_sig,
    READY=>READY4,
    divA=>DA_sig,
    divB=>DB_sig);

Divider_U: Divider
    port map(
    CLK=>CLK2,
    RST=>RST,
    LOAD=>SIG4_5,
    divA=>DA_sig,
    divB=>DB_sig,
    READY=>READY,
    U=>U);


--Infer-Defuzz
process(CLK2,RST)
begin
if RST='1' then
    SIG2_4<='0';
```

```
elsif CLK2'event and CLK2='1' then
  if READY2='1' then
     SIG2_4<='1';
  else
     SIG2_4<='0';
  end if;
end if;
end process;

--Defuzz-Divider
process(CLK2,RST)
begin
if RST='1' then
     SIG4_5<='0';
elsif CLK2'event and CLK2='1' then
  if READY4='1' then
     SIG4_5<='1';
  else
     SIG4_5<='0';
  end if;
end if;
end process;

--Probe
--x1<=x1_sig;
--x2<=x2_sig;

end Control_arc;

--configuration Control_conf1 of Control is
--for Control_arc
--    for Deriv_U: Deriv use configuration work.Deriv_conf1;
--    end for;
--    for Fuzzify2_U: Fuzzify2 use configuration work.Fuzzify2_conf1;
--    end for;
--    for Infer_U: Infer use configuration work.Infer_conf1;
--    end for;
--    for Defuzz_U: Defuzz use configuration work.Defuzz_conf1;
--    end for;
--    for Divider_U: Divider use configuration work.Divider_conf1;
--    end for;
--end for;
--end Control_conf1;
```

Appendix C

PWM Controllers

App. C-1 : C++ Program to generate PWM waveform

```
// PWM Waveform Generator
// C++
// Revised Version : 11 Feb 1998
//
// This program generates the PWM waveforms
// based on the desired parameters.

#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <math.h>
#include <stdio.h>

// Function prototype
int Sign(float);

int main(void)
    {

    fstream out_file;          //define stream object

    char filename[30] ;
    char T;
    int Choice;
    float n,NT,M,N;

    //initialise values
    N=4096;    //EPROM memory spaces, Sinewave period


    int A[4096],B[4096],C[4096],NA[4096],Out[4096];
    float Tri[4096];

    cout <<"\n    Pulse Width Modulation Pattern Generation Program";
    cout <<"\n                    by";
    cout <<"\n                 Jeen G. Khor";


    // Request for parameters
    cout <<"\n";
    cout <<"\n1. Three Phase.";
    cout <<"\n2. Single Phase, bipolar voltage switching.";
    cout <<"\n3. Single Phase, half controlled switching.";
    cout <<"\n\nEnter selection (1-3): ";
    cin >>Choice;
    cout <<"\nEnter triwave period, NT (N=4096) : ";
    cin >>NT;
    cout <<"Enter amplitude modulation factor : ";
    cin >>M;
```

```
//Triangular wave generation
Tri[0]=0;
for(n=1;n<N;n++)
    {
    Tri[n]=Tri[n-1]+(Sign(sin(2*M_PI*(n/NT)+M_PI)-sin(2*M_PI*((n-1)/NT)+M_PI))*(4.0/NT));
    }

//Comparator - Phase A only
for(n=0;n<N;n++)
    {
    // Phase A
    if((M*sin(2*M_PI*(n/N))) >= Tri[n])
        {
        A[n]=2;
        NA[n]=1;
        }
    else
        {
        A[n]=1;
        NA[n]=2;
        }
    }

// Defining Phases B & C - shift by 120deg (1365)
for (n=0;n<N;n++)
    {
    // Phase B
    if((M*sin((2*M_PI*(n/N))+2.094395)) >= Tri[n])
        {
        B[n]=2;
        }
    else
        {
        B[n]=1;
        }

    // Phase C
    if((M*sin((2*M_PI*(n/N))-2.094395)) >= Tri[n])
        {
        C[n]=2;
        }
    else
        {
        C[n]=1;
        }
    }


// Assigning values for Out[n]

switch(Choice) {
    case 1:
        for (n=0;n<N;n++)
            {
            Out[n]= A[n]+(B[n]*4)+(C[n]*16);
            }
        break;
    case 2:
        for (n=0;n<N;n++)
            {
            Out[n]= A[n]+(NA[n]*4);
            }
        break;
```

```
case 3:
    for (n=0;n<N;n++)
    {
        if (n<(N/2))
            {
            Out[n]=A[n]+(2*4);
            }
        else
            {
            Out[n]=A[n]+(1*4);
            }

        }
    }
/***** Filing Operation *****/

cout <<"\nEnter name of input file : ";
gets(filename);
cout <<"\nInclude test parameters (y/n) ? ";
cin >>T;
out_file.open(filename, ios::out);

// Filling Error
if(! out_file)
    {
    cout << "\nUnable to open file ";
    return 1;
    }

cout << "\nWriting data into file   : " << filename;

// Writing into file
if(T=='y'|| T=='Y')
    {
    out_file <<"<Testing Parameters>\n";

    switch(Choice)
        {
        case 1:
            out_file <<"Three Phase\n";
            break;
        case 2:
            out_file <<"Single Phase (full)\n";
            break;
        case 3:
            out_file <<"Single Phase (half)\n";
            break;
        }
    out_file <<"N=" <<N <<" ; NT=" <<NT <<" ; Ma="<<M <<"\n\n";
    }

for(n=0;n<N;n++)
    {
    out_file << hex << Out[n] <<" ";
    }

    out_file.close();

    cout <<"\n\nOk.";

    return 0;
    }

/***** End of Main() *****/
```

```
//FUNCTION : Sign
//PURPOSE  : To return the sign of the float a.
int Sign(float a)
     {
     if(a<0)
          {
          return -1;
          }
     else
          {
          return 1;
          }
}
```

App. C-2 : PIC Assembly code to control SA828

```
;This programs initialises the SA828 pulse width modulation chip

LIST C=80, P=16C84, F=INHX8M
      include "c:\pic\p16cxx.inc"
      list

;************************ EQUATES ***************************
;*********************PIN/BIT DEFINITIONS************************
      __FUSES _CP_OFF&_PWRTE_OFF&_WDT_OFF&_XT_OSC
;       __CONFIG  11H

 #DEFINE ALE    0          ;PORT A
 #DEFINE WRTE   1          ;PORT A
 #DEFINE RST    2          ;PORT A
 #DEFINE LIGHT  3     ;PORT A CONFIDENCE LIGHT
 #DEFINE SWTCH  4     ;PORT A INPUT

 ;PORTB IS ALL ADDRESS LINES AND DATA LINES

 ADDRSS      EQU   11H
 DAT         EQU   12H
 COUNTER     EQU   13H
 COUNTER2    EQU   14H

 ORG    000H
 RESET  GOTO   START


;************************MAIN PROGRAM*****************************
START   CLRF   STATUS        ;Initialise port b as outputs
      MOVLW   0X00
      MOVWF   PORTB
      MOVLW   0X00
      TRIS    PORTB
```

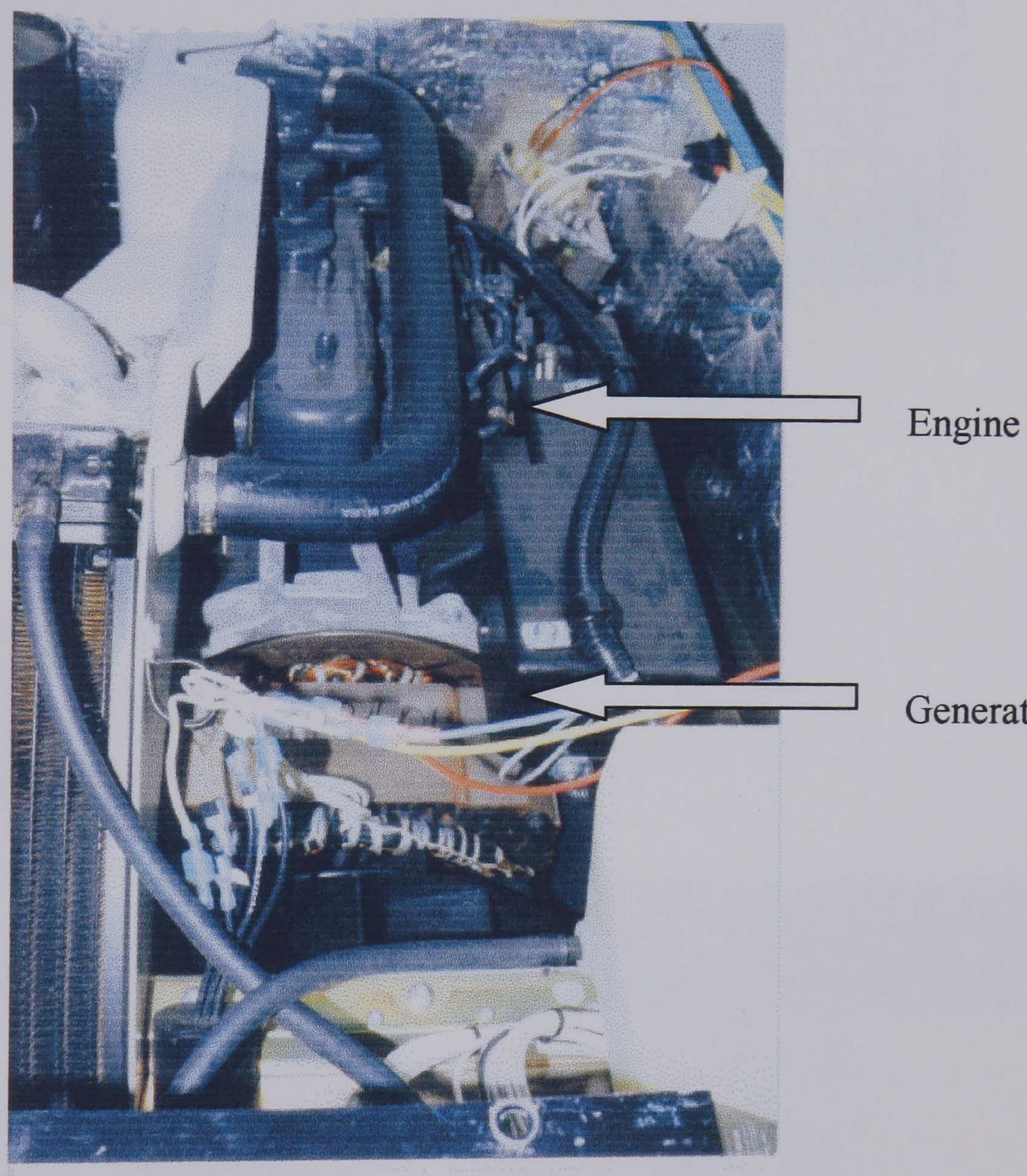
```
        ; port a arranged as all outputs
        CLRF    PORTA
        MOVLW   0X10
        TRIS    PORTA

        BCF     EEADR,7         ;Clear EE top addresses to minimise power
        BCF     EEADR,6

;SORT OUT ALL THE INTERRUPTS
        BCF     INTCON,GIE              ;GLOBAL INTERRUPT DISABLE
        BCF     INTCON,EEIE             ;NO EEPROM INTERRUPT
        BCF     INTCON,T0IE             ;NO TIMER INTERRUPT
        BSF     INTCON,INTE             ;PORT B PIN 6 INTERRUPT ENABLED
        BCF     INTCON,RBIE             ;CHANGE ON PORT B INTERRUPT BISABLED
        BSF     STATUS,RP0
        BCF     0X1,INTEDG              ;INTERRUPT ON FALLING EDGE
                                            ;FOR PORT B PIN 6 INTERRUPT
        BCF     STATUS,RP0


;****FIRST MAKE SURE DEVICE IS RESET**************
        BCF     PORTA,RST
MAIN
                        ;*******SENDS D2 TO ADRESS 0
        MOVLW   0
        MOVWF   ADDRSS
        MOVLW   0XD2
        MOVWF   DAT
        CALL    SENDIT

                    ;*******SENDS 0 TO ADRESS 1
        MOVLW   1
        MOVWF   ADDRSS
        MOVLW   0X00
        MOVWF   DAT
        CALL    SENDIT

                    ;*******SENDS 7F TO ADRESS 2
        MOVLW   2
        MOVWF   ADDRSS
        MOVLW   0X7F
        MOVWF   DAT
        CALL    SENDIT

                    ;*******SENDS FF TO ADRESS 4
        MOVLW   4
        MOVWF   ADDRSS
        MOVLW   0XFF
        MOVWF   DAT
        CALL    SENDIT
                    ;*******SENDS CD TO ADRESS 0
        MOVLW   0
        MOVWF   ADDRSS
        MOVLW   0XCD
        MOVWF   DAT
        CALL    SENDIT
                    ;*******SENDS 0C TO ADRESS 1
        MOVLW   1
        MOVWF   ADDRSS
        MOVLW   0X0C
        MOVWF   DAT
        CALL    SENDIT

                    ;*******SENDS CC TO ADRESS 2
```

```
MOVLW  2
MOVWF  ADDRSS
MOVLW  0XCC
MOVWF  DAT
CALL   SENDIT

                ;*******SENDS FF TO ADRESS 3
MOVLW  3
MOVWF  ADDRSS
MOVLW  0XFF
MOVWF  DAT
CALL   SENDIT

;Enable PWM Output
    BSF    PORTA,RST


                ;*******SENDS CD TO ADRESS 0
    MOVLW  0
    MOVWF  ADDRSS
    MOVLW  0XCD
    MOVWF  DAT
    CALL   SENDIT
                ;*******SENDS 2C TO ADRESS 1
    MOVLW  1
    MOVWF  ADDRSS
    MOVLW  0X2C
    MOVWF  DAT
    CALL   SENDIT

                ;*******SENDS CC TO ADRESS 2
MOVLW  2
MOVWF  ADDRSS
MOVLW  0XCC
MOVWF  DAT
CALL   SENDIT

                ;*******SENDS FF TO ADRESS 3
MOVLW  3
MOVWF  ADDRSS
    MOVLW  0XFF
    MOVWF  DAT
    CALL   SENDIT


WAIT1  MOVLW  0XFF
    MOVWF  COUNTER
    MOVWF  COUNTER2

WAIT2  NOP
    NOP
    DECFSZ COUNTER,F
    GOTO   WAIT2
    DECFSZ COUNTER2,F
    GOTO   WAIT2
    BSF    PORTA,LIGHT

    MOVLW  0XFF
    MOVWF  COUNTER
    MOVWF  COUNTER2
WAIT3  NOP
    NOP
    DECFSZ COUNTER,F
    GOTO   WAIT3
```

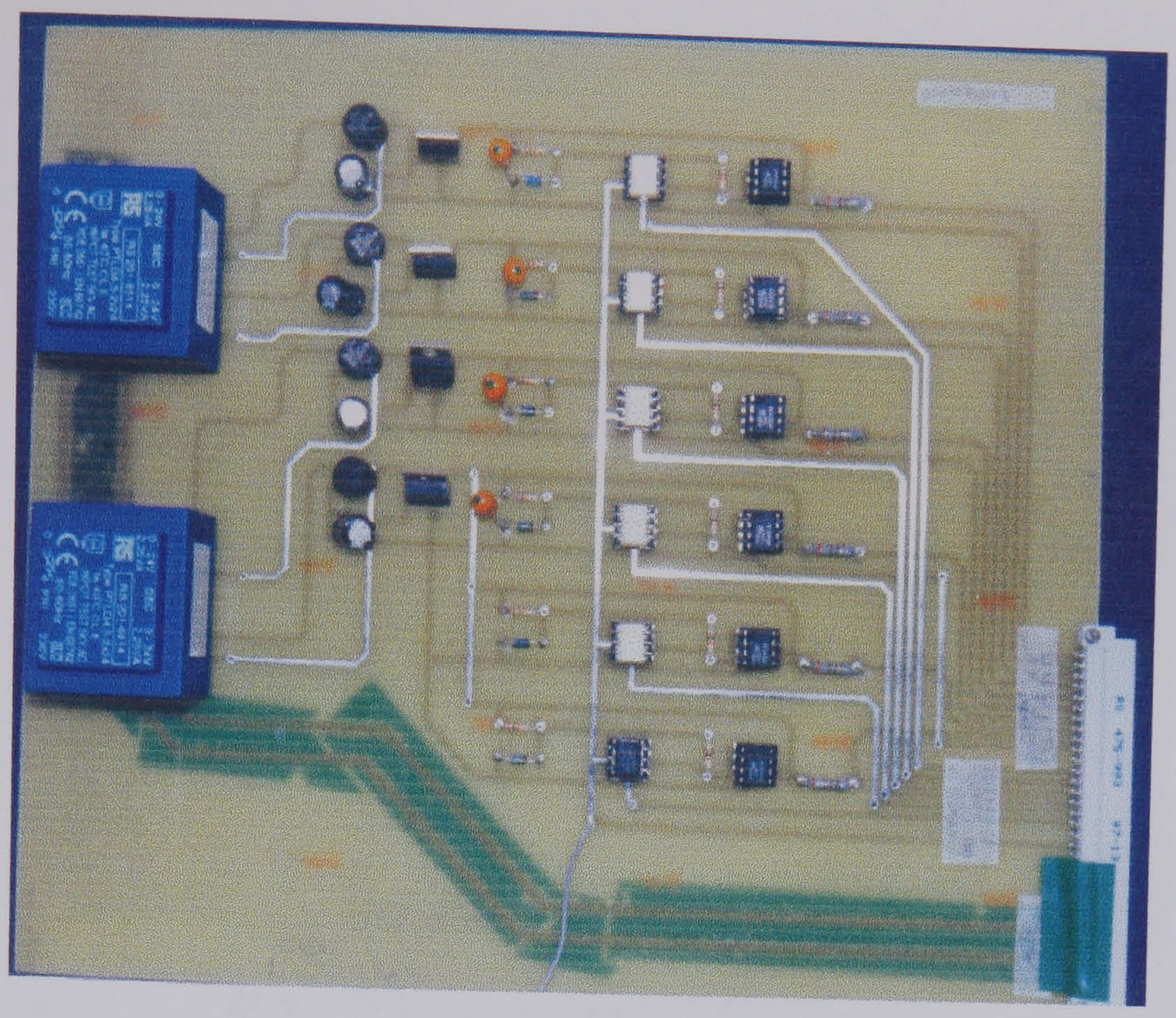
```
        DECFSZ  COUNTER2,F
        GOTO    WAIT3

        BCF     PORTA,LIGHT

        GOTO    WAIT2

        GOTO    MAIN

;THIS SUBROUTINE IS USED TO DRIVE ALE AND WRTE HIGH AND LOW
;AS PER THE INTEL TIMING SPECIFICATIONS

SENDIT
        BSF     PORTA,ALE       ;take ale high
        MOVF    ADDRSS,W        ;send the address
        MOVWF   PORTB
        BCF     PORTA,ALE       ;take ale low

        BCF     PORTA,WRTE      ;take write low
        MOVF    DAT,W
        MOVWF   PORTB           ;send the data
        BSF     PORTA,WRTE      ;take wrte high
        RETURN

 END
```

Appendix D

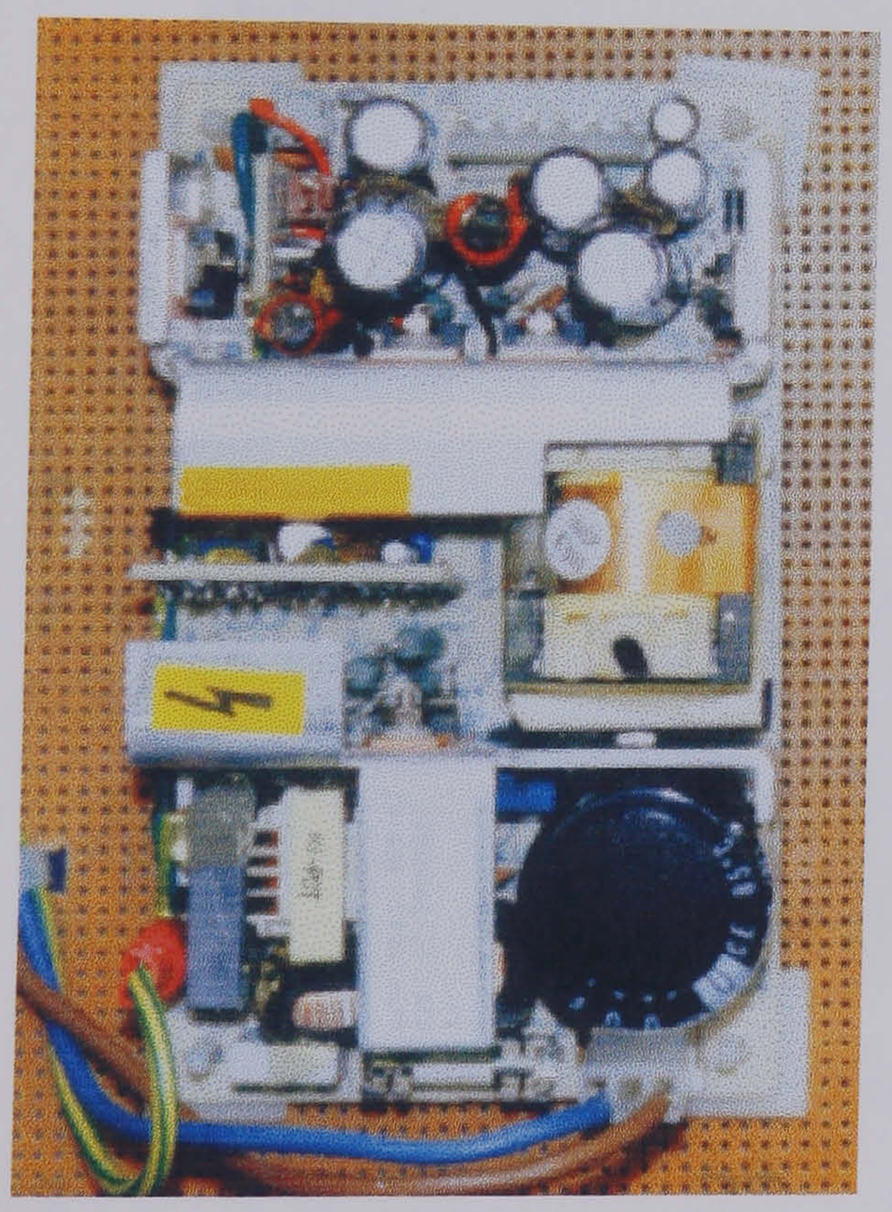
Hardware



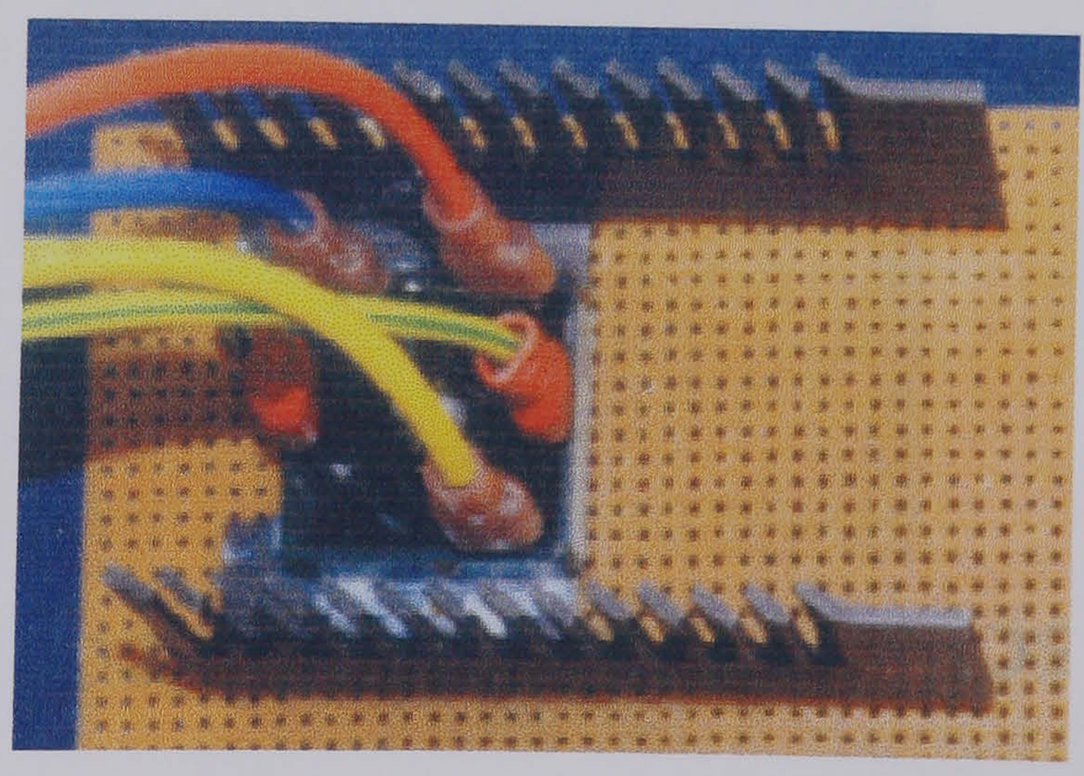
D-1 *Diesel engine and permanent magnet synchronous generator.*



D-2 *Driver board for IGBT inverter*

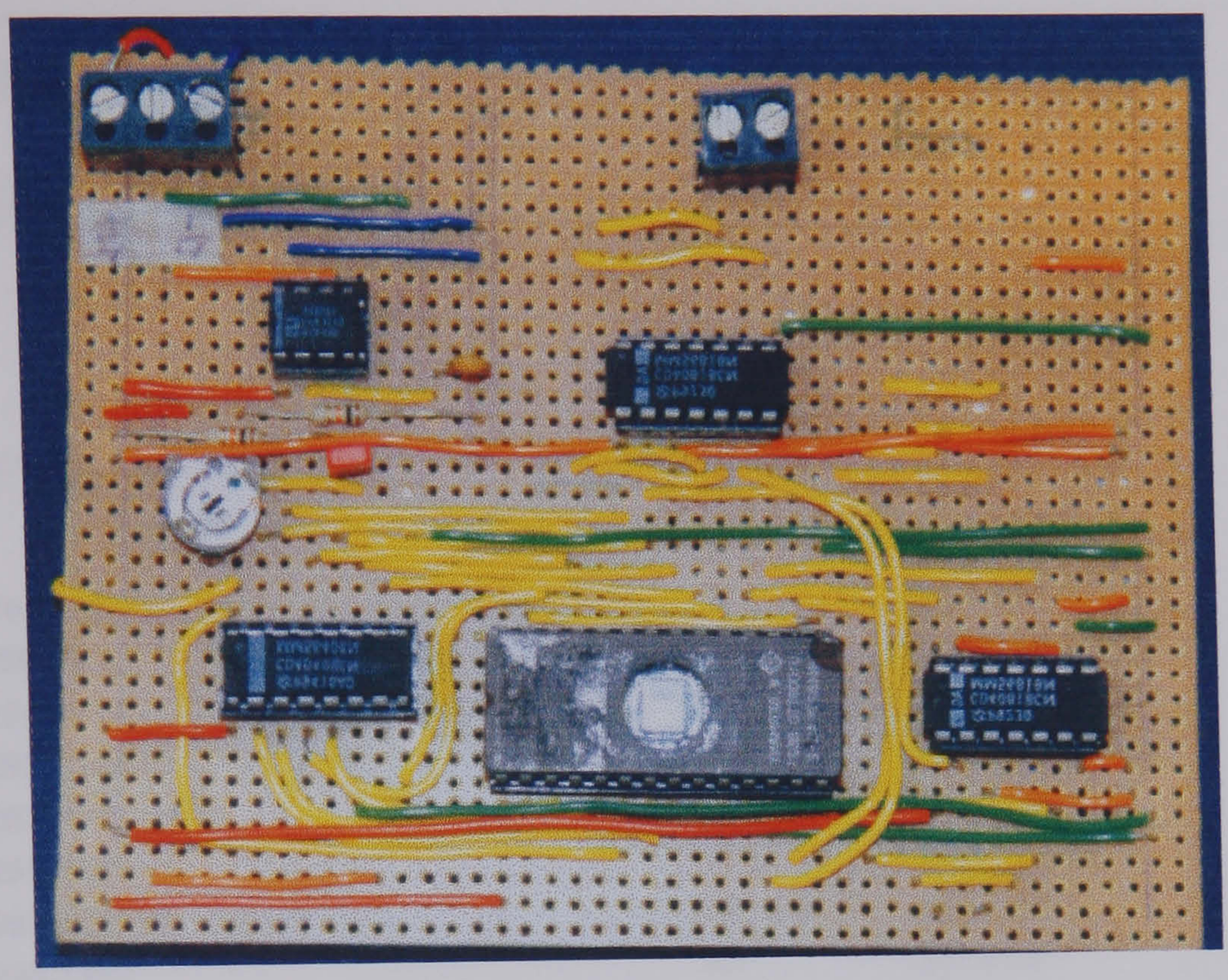


(a)

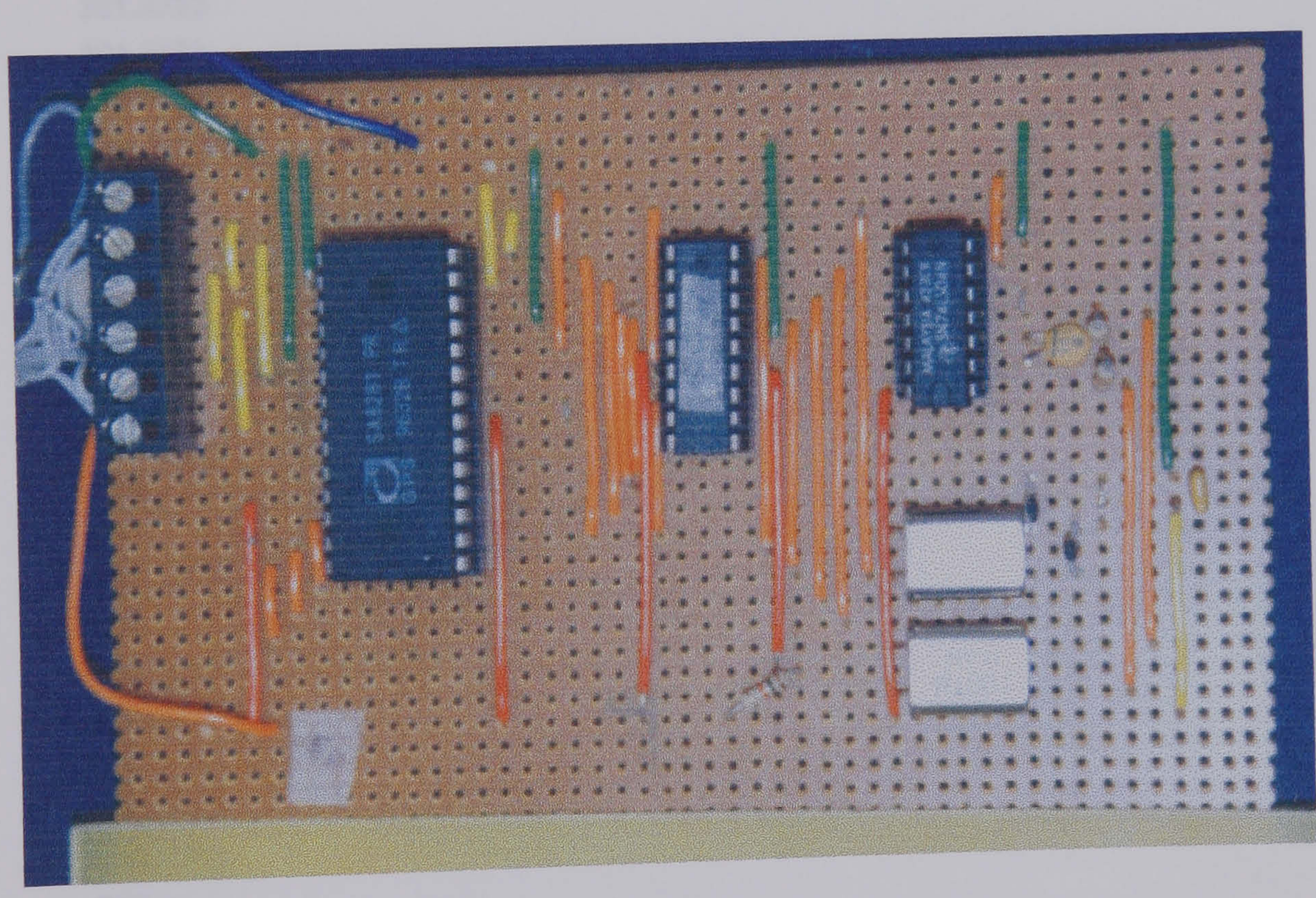


(b)

D-3 *(a)5v-12v Power supply unit (b)Three phase uncontrolled rectifier.*



D-4 *PWM controller (EPROM circuit).*



D-5 *PWM Controller (SA828 circuit).*

Appendix E

Test Results

Data for Figure 8-16

Time [sec]	**Voltage**
0	**296**
1	**294.6667**
2	**293.3333**
3	**293.3333**
4	**293.3333**
5	**293.3333**
6	**293.3333**
7	**294.6667**
8	**293.3333**
9	**297.3333**
10	**301.3333**
11	**301.3333**
12	**301.3333**
13	**246.6667**
14	**197.3333**
15	**189.3333**
16	**269.3333**
17	**356**
18	**258.6667**
19	**173.3333**
20	**126.6667**
21	**136**
22	**240**
23	**349.3333**
24	**352**
25	**314.6667**
26	**260**
27	**213.3333**
28	**209.3333**
29	**253.3333**
30	**296**
31	**312**
32	**285.3333**
33	**253.3333**
34	**230.6667**
35	**220**
36	**233.3333**
37	**270.6667**
38	**294.6667**
39	**302.6667**
40	**285.3333**
41	**269.3333**
42	**261.3333**
43	**257.3333**
44	**254.6667**
45	**253.3333**
46	**253.3333**
47	**252**
48	**253.3333**
49	**253.3333**
50	**256**
51	**260**

Data for Figure 8-17

time [sec]	**voltage**	*17*	**244**	*35*	**272**
0	**282.6667**	*18*	**262.6667**	*36*	**269.3333**
1	**282.6667**	*19*	**270.6667**	*37*	**269.3333**
2	**282.6667**	*20*	**269.3333**	*38*	**266.6667**
3	**282.6667**	*21*	**269.3333**	*39*	**266.6667**
4	**280**	*22*	**266.6667**	*40*	**266.6667**
5	**281.3333**	*23*	**268**	*41*	**266.6667**
6	**286.6667**	*24*	**266.6667**	*42*	**266.6667**
7	**288**	*25*	**268**	*43*	**268**
8	**288**	*26*	**266.6667**	*44*	**268**
9	**286.6667**	*27*	**265.3333**	*45*	**268**
10	**288**	*28*	**266.6667**	*46*	**268**
11	**286.6667**	*29*	**266.6667**	*47*	**268**
12	**284**	*30*	**266.6667**	*48*	**266.6667**
13	**265.3333**	*31*	**268**	*49*	**268**
14	**250.6667**	*32*	**269.3333**	*50*	**270.6667**
15	**228**	*33*	**269.3333**	*51*	**269.3333**
16	**222.6667**	*34*	**269.3333**		

Data for Figure 8-18

Time [sec]	**voltage**	*15*	**242.6667**	*32*	**0**
0	**246.6667**	*16*	**241.3333**	*33*	**0**
1	**245.3333**	*17*	**242.6667**	*34*	**0**
2	**245.3333**	*18*	**241.3333**	*35*	**0**
3	**245.3333**	*19*	**242.6667**	*36*	**0**
4	**245.3333**	*20*	**238.6667**	*37*	**0**
5	**244**	*21*	**238.6667**	*38*	**0**
6	**241.3333**	*22*	**228**	*39*	**0**
7	**237.3333**	*23*	**189.3333**	*40*	**0**
8	**238.6667**	*24*	**152**	*41*	**0**
9	**237.3333**	*25*	**130.6667**	*42*	**0**
10	**237.3333**	*26*	**117.3333**	*43*	**0**
11	**238.6667**	*27*	**90.66667**	*44*	**0**
12	**241.3333**	*28*	**37.33333**	*45*	**0**
13	**242.6667**	*29*	**1.333333**	*46*	**0**
14	**242.6667**	*30*	**0**		
		31	**0**		

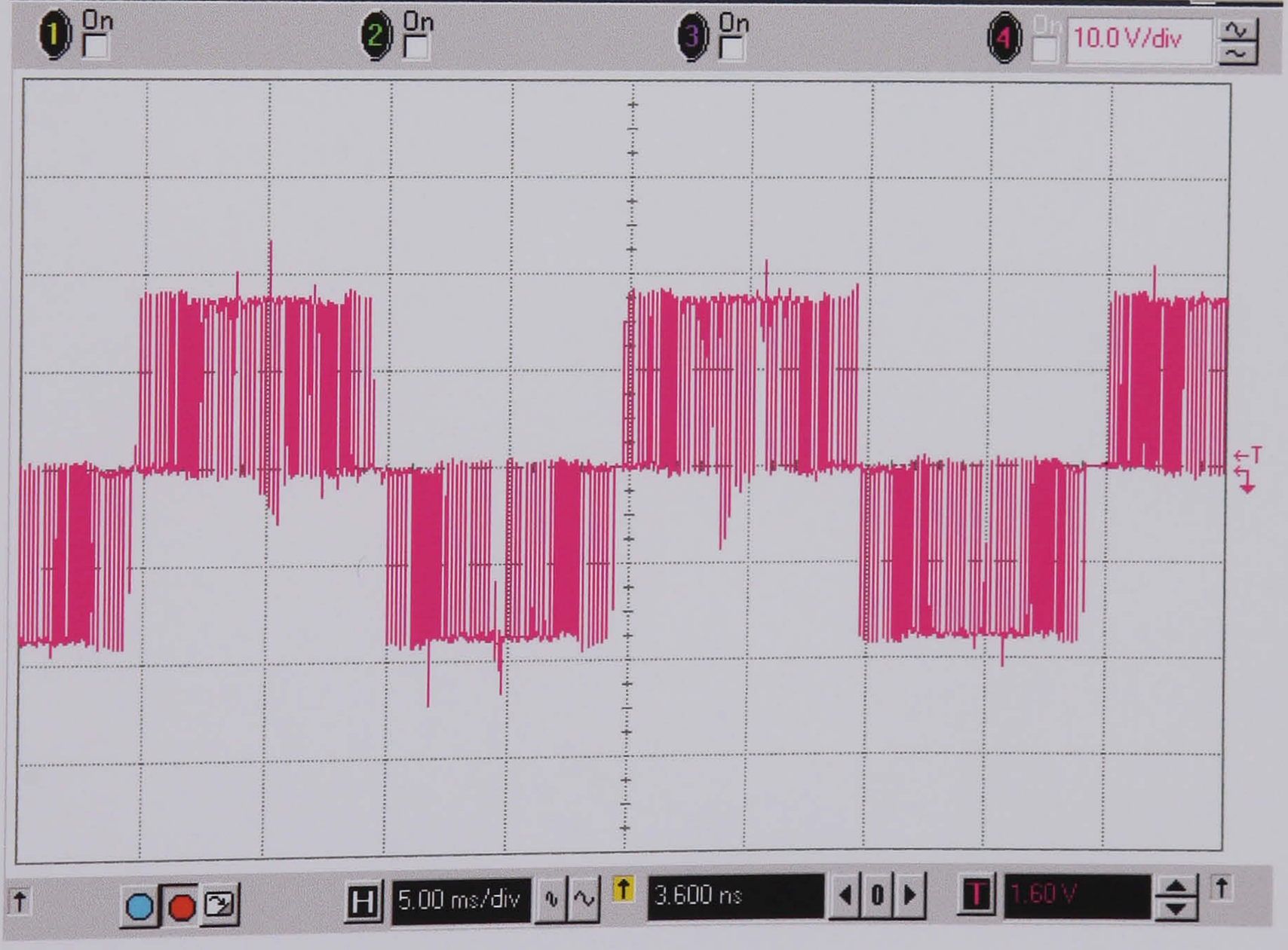
Data for Figure 8-19

Time [sec]	**voltage**	*15*	**261.3333**	*31*	**256**
0	**266.6667**	*16*	**260**	*32*	**256**
1	**266.6667**	*17*	**260**	*33*	**253.3333**
2	**266.6667**	*18*	**260**	*34*	**254.6667**
3	**265.3333**	*19*	**262.6667**	*35*	**254.6667**
4	**264**	*20*	**262.6667**	*36*	**256**
5	**262.6667**	*21*	**262.6667**	*37*	**256**
6	**262.6667**	*22*	**257.3333**	*38*	**257.3333**
7	**262.6667**	*23*	**244**	*39*	**258.6667**
8	**261.3333**	*24*	**230.6667**	*40*	**258.6667**
9	**261.3333**	*25*	**222.6667**	*41*	**257.3333**
10	**260**	*26*	**230.6667**	*42*	**256**
11	**261.3333**	*27*	**249.3333**	*43*	**256**
12	**262.6667**	*28*	**257.3333**	*44*	**254.6667**
13	**264**	*29*	**260**	*45*	**256**
14	**264**	*30*	**257.3333**	*46*	**256**

Data for Figure 8-21

time [sec]	**voltage**	*2*	**278.6667**	*29*	**258.6667**
0	**280**	*3*	**277.3333**	*30*	**254.6667**
1	**278.6667**	*4*	**276**	*31*	**252**
2	**278.6667**	*5*	**277.3333**	*32*	**252**
3	**278.6667**	*6*	**276**	*33*	**250.6667**
4	**277.3333**	*7*	**276**	*34*	**250.6667**
5	**278.6667**	*8*	**276**	*35*	**252**
6	**277.3333**	*9*	**274.6667**	*36*	**252**
7	**278.6667**	*10*	**274.6667**	*37*	**252**
8	**278.6667**	*11*	**276**	*38*	**253.3333**
9	**277.3333**	*12*	**274.6667**	*39*	**250.6667**
10	**277.3333**	*13*	**250.6667**	*40*	**252**
11	**273.3333**	*14*	**233.3333**	*41*	**253.3333**
12	**273.3333**	*15*	**228**	*42*	**253.3333**
13	**273.3333**	*16*	**233.3333**	*43*	**256**
14	**277.3333**	*17*	**240**	*44*	**256**
15	**277.3333**	*18*	**245.3333**	*45*	**256**
16	**277.3333**	*19*	**245.3333**	*46*	**253.3333**
17	**277.3333**	*20*	**252**	*47*	**250.6667**
18	**277.3333**	*21*	**253.3333**	*48*	**250.6667**
19	**278.6667**	*22*	**254.6667**	*49*	**250.6667**
20	**280**	*23*	**256**	*50*	**248**
21	**281.3333**	*24*	**256**	*51*	**252**
22	**280**	*25*	**258.6667**	*52*	**252**
23	**278.6667**	*26*	**257.3333**	*53*	**250.6667**
0	**278.6667**	*27*	**256**		
1	**277.3333**	*28*	**258.6667**		

***E-1** PWM Inverter Output: Phase Voltage*



***E-2** PWM Inverter Output: Line Voltage*

E-3 PWM Inverter Output: Line Currents